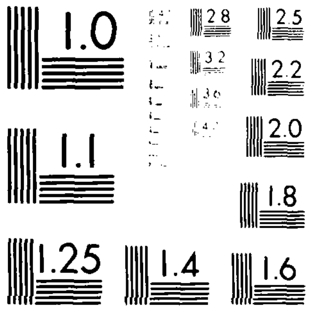
AD-A142 570 WIS IMPLEMENTATION STUDY REPORT VOLUME 3 BACKGROUND INFORMATION(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T H PROBERT 01 OCT 83 IDA-D-51-VOL-3 1/6

UNCLASSIFIED IDA/HQ-84-28344 MDA903-79-C-0018 F/G 17/2 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Copy 85 of 128 copies

2

AD-A142 570

IDA RECORD DOCUMENT D-51

WIS IMPLEMENTATION STUDY REPORT-- VOLUME III-- BACKGROUND INFORMATION

Thomas H. Probert, *Project Leader*

October 1, 1983

Prepared for
Office of the Under Secretary of Defense for Research and Engineering

IDA

INSTITUTE FOR DEFENSE ANALYSES

84 06 27 084

IDA Log No. HQ84-28344

The work reported in this document was conducted under contract MDA 903 79 C 0018 for the Department of Defense. The publication of this IDA Record Document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. ADA142576	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) WIS Implementation Study Report-- Volume III--Background Information		5. TYPE OF REPORT & PERIOD COVERED Final -- September 1983
		6. PERFORMING ORG. REPORT NUMBER IDA Record Document D-51
7. AUTHOR(s) Thomas H. Probert, Project Leader		8. CONTRACT OR GRANT NUMBER(s) MDA 903 79 C 0018
9. PERFORMING ORGANIZATION NAME AND ADDRESS Institute for Defense Analyses 1801 N. Beauregard Street Alexandria, VA 22311		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task T-4-206
11. CONTROLLING OFFICE NAME AND ADDRESS JPM WIS, Director, Technology Directorate 7798 Old Springhouse Road McLean, VA 22102		12. REPORT DATE October 1, 1983
		13. NUMBER OF PAGES 517
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DoD-IDA Management Office 1801 N. Beauregard Street Alexandria, VA 22311		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE NA

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is the result of a workshop conducted by the Institute for Defense Analyses to develop the functional specifications and estimates of implementation effort for foundation building blocks for Command and Control Systems in WIS. The group concluded that: 1) the development of a modernized WIS incorporating the specified foundation technology building blocks can be accomplished within the time frame proposed, 2) the use of Ada and proposed

(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Continued

information processing standards are appropriate for use in WIS modernization and are expected to reduce the time required to implement the full system, and 3) it is critical to the success of the WIS modernization that major attention be paid to interface definition and design, system integration and test, and configuration management of the system while under development.

IDA RECORD DOCUMENT D-51

WIS IMPLEMENTATION STUDY REPORT-- VOLUME III-- BACKGROUND INFORMATION

Thomas H. Probert, *Project Leader*

October 1, 1983

IDA

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

Contract MDA 903 79 C 0018
Task T-4-206

FOREWORD

On June 10, 1983, Dr. Richard DeLauer, Under Secretary of Defense for Research and Engineering, approved the final draft of Department of Defense Directive (DoDD) 5000.31, "Computer Programming Language Policy," and circulated it for coordination. This directive establishes Ada as the single common high order language for Defense mission-critical applications. The World-Wide Military Command and Control System (WWMCCS) is specifically identified in supporting documentation to that directive as a mission-critical computer application.

In anticipation of final approval of this directive, the WWMCCS Information System Joint Program Management Office (WIS JPMO) requested the Institute for Defense Analyses to undertake a project to develop the functional specifications and estimates of implementation effort for foundation building blocks for Command and Control Systems. These software capabilities will be used to support the operation of the WIS and will be developed in Ada.

These eleven key foundation building blocks have been divided into two groups: near-term areas for which the specified packages will be operational by January 1986 and mid-term areas for which the specified packages would be operational by January 1989. Near-term areas are characterized by encompassing mature technology that is currently embodied in operational systems. Development of packages for these areas should capitalize on existing software requirement definitions and design specifications. Mid-term areas are characterized as or near. the current state of the art and will require significant requirements analysis, architecture and design specification activities.

The participants in this analysis, specification and planning study were chosen according to three criteria: they are all recognized experts in respective key technical areas, they all have had direct implementation experience, and they were all chosen with regard to broad representation of the technical and commercial community. Background information for these people can be found in Volume II of this Record Document.

The study was performed in four phases. First, individual experts were selected for their recognized expertise in each of the foundation areas. One expert in each area, working independently, was tasked to produce a working paper describing the state of the art, discussing the technical issues, and venturing predictions for possible extensions to that state of the art. Each report presents an overview, a discussion of functional requirements for the system or for a set of packages for the system, case studies dealing with similar existing systems, analysis of the information including cost and schedule estimates, and conclusions. In the mid-term areas the case studies were replaced by discussion of the state of the art, the state of practice, and forecasts of new products. These reports were collected and distributed to the larger group of selected experts according to their expertise as preparation for the cluster workshops. These reports can be found in Volume III of this report.

In the second phase, four "cluster" workshops were conducted to perform similar analysis for each foundation area. These workshops addressed these foundation areas grouped according to technical similarity and dependence. Each cluster workshop used the submitted expert assessment as a baseline and point of departure. The goal of these cluster workshops was to assess the content of the reports and make recommendations regarding consistency, bias, level of effort, etc., such that the contents of these reports and the workshops could be merged into a final document.

The third phase entailed the analysis of all the cluster workshop reports in a *meeting of the cluster chairpersons*. This was conducted to eliminate areas of redundancy and assess the effort required to integrate all the foundation areas into a coherent system description and estimate of total effort. These conclusions can be found in the Executive Summary, Volume I of this report.

Finally, this Record Document has been prepared by the IDA project management, cluster chairpersons and Computer and Software Engineering Division technical staff.

Thomas H. Probert
Project Leader

CONTENTS -- Volume III

CONTENTS -- Volume I

CONTENTS -- Volume II

PRECEDING PAGE BLANK-NOT FILMED

CLUSTER I PAPERS

TECHNICAL REPORT: OPTIMIZATION TECHNIQUES AND ARTIFICIAL INTELLIGENCE METHODOLOGIES

21 September 1983

TITAN SYSTEMS, INC. 6833 Beverly Road, Suite 320, McLean, Virginia 22101 (703) 790-1120

TABLE OF CONTENTS

SECTION 1
OVERVIEW

The purpose of this report is to support an effort by the Institute for Defense Analysis that is developing estimates of the cost and scope of component subsystems for the next generation World Wide Military Command Control System (WWMCCS). This report will review and summarize optimization techniques and explore the application of Artificial Intelligence (AI) methodologies to improve material and transportation planning.

1.1 INTRODUCTION

Problem solving systems can usually be described in terms of three main components. The first of these is a database which describes both the current task-domain situation and the goal. Dependent upon the application, the database can be comprised of a variety of data structures including maps, lists, property list structures, and semantic networks. In theorem proving, for example, the current task-domain situation consists of assertions representing axioms, lemmas, and theorems already proven; the goal is an assertion representing the theorem to be proved. In robot problem solving, a current situation is a world model consisting of statements describing the physical surroundings of the robot, and the goal is a description that is to be made true by a sequence of robot actions.

The second component of problem-solving systems is a set of operators that are used to manipulate the database. Examples of typical operators include rules for moving chessmen, rules of inference for theorem proving and rules for simplifications of mathematical integration techniques.

The third component of a problem solving system is a control strategy for determining the sequence of events -- i.e. when and where to apply a particular operator.

In general, the objective is to achieve a goal through application of a sequence of operators to an initial task-domain situation. The application of the operators to the database structures to produce a modified task-domain situation is then called _reasoning forward_. Alternatively, _reasoning backward_ involves the application of an operator to the _goal_ to produce subgoals and sub-subgoals that are easier to solve. An important technique involving both forward and backward reasoning is called _means-ends analysis_. This involves comparing the current goal with a current task-domain situation in order to determine the difference between them. This difference is then used to select the most suitable operator to reduce the difference.

1.2 STATE SPACES AND PROBLEM REDUCTION

A problem-solving system that uses forward reasoning and whose operators each work by producing a single new state in the database is said to represent problems in a _state-space representation_.

For backward reasoning, a distinction may be drawn between two cases. In one, each application of an operator to a problem yields exactly one new problem, whose size or difficulty is typically slightly less than that of the previous problem. Systems of this kind will also be referred to as employing state-space representations.

The second case occurs when backward reasoning results in a set of subproblems, each significantly smaller than the original. A system where backward reasoning changes a single object into a conjunction of objects is said to employ a _problem-reduction representation_.

In addition to the state-space and problem - reduction approaches, there are a number of other variations of problem representation. One of these is the _game tree_ which represents a game playing problem in a manner which takes into account the potential adversary moves.

1.3 GRAPH REPRESENTATION

In either a state-space or problem-reduction representation, achieving the desired goal can be equated with finding an appropriate finite sequence of applications of available operators. In this context, we define search as the methodology for determining the appropriate operator sequence.

Tree structures are commonly used in implementing control strategies for the search. In a state-space representation, a tree may be used to represent the set of problem states produced by operator applications. In such a representation, the root node of the tree represents the initial problem situation or state. Each of the new states that can be produced from this initial state by the application of just one operator is represented by a successor node of the root node. Subsequent operator applications produce successors of these nodes, and so on. Each operator application is represented by a directed arc of the tree. More generally, this is represented in a graph rather than a tree, since there can be numerous paths between given nodes. For applications involving problem-reduction, AND/OR graphs provide a means of tracking the subgoals attempted and the subgoal aggregation to achieve the original goal.

1.4 SEARCH SPACE

The objective of achieving a state that satisfies the goal condition can now be formulated as the problem of searching a graph to find a particular node that satisfies the objective state. The search space consists of all alternative nodes with paths from the initial state. Many problem domains have an infinite or near infinite search space. Checkers, for example, has a search space estimated at 10^{40}.

The critical issue now becomes the amount of time required to find a suitable solution, given a particular search space.

Several graph and tree searching methods have been developed and they play an important role in the control of problem-solving processes. Among these are: Blind state-space search; Blind AND/OR graph search; and Game tree search, including Minimax, and Alpha-beta pruning procedures.

Of particular interest are those graph-searching methods that use heuristic knowledge from the problem domain to help narrow the search. Heuristic search techniques have proven to be one of the key contributions of AI to efficient problem solving. A number of heuristic search techniques utilize knowledge to focus and minimize the search. Among these are: Heuristic state-space search; Heuristic AND/OR graph search, A*-Optimal and bidirectional searches.

1.5 PLANNING

An alternative to the problem of limiting search, is to have the problem-solving system find a better representation automatically. The STRIPS[1] program made initial advances in this area by augmenting its initial set of operators by generating macro-operators, based on problem-solving experience. ABSTRIPS[2] makes further advances by filling in the detailed solution only after a satisfactory outline of the solution (or plan) has been found.

1.6 AI APPLICATON

The use of AI search and planning methodologies as described above have a number of potential applications to the material/transportation planning process. These will be discussed further in Section 2.

[1]Written by Richard Fikes and Nils Nilsson at SRI International (1971)

[2]Implemented by Earl Sacerdoti (1974)

SECTION 2
FUNCTIONAL REQUIREMENT

2.1 AI APPLICATION TO MATERIAL/TRANSPORTATION PLANNING

Table 2-1 lists a number of areas in which AI methodologies could be applied in a beneficial manner to enhance the flexibility, efficiency, and utility of several functional elements required for the implementation of a material/transportation planning system.

TABLE 2-1

APPLICATION DESCRIPTION	PURPOSE
Heuristic Models - Automated Decision Aid - Expert Systems	Provides the capability for utilizing the knowledge base of an "expert(s)" to structure the decision rules. Provides the capability for human interaction to dynamically restructure and interface with the decision-making process.
Heuristic Optimization Techniques	Provides potential for simplified alternative to exact optimization approach in areas such as application specific models and simulations; Network model representation.
Data Base Management	Capabilities such as relational data base, semantic networks and frame architectures afford opportunity to provide inference data chains; textually intensive applications have potential for hierarchical inference (ZOG, BROWZING)
Generalized Planning Systems	Hierarchical, Scripted, and Opportunistic planning programs to model individual components and modules of the material/ transportation planning process.

2.2 TECHNICAL CHALLENGES

The implementation of a system utilizing the AI tools previously described has a number of areas of concern which must be adequately satisfied to achieve a successful operational capability. These include:

(1) Objective Definition

The definition of the goal and the formulation of the objective function is necessarily dependent upon a number of parameters. The commercial sector is most often driven by the goal of profit maximization. The military, however, may be faced with a scenario dependent objective. This is not a prohibitive concern, as always the objective function is multivariate; however, there is a need to emphasize the subproblem interaction analysis to assure that any conflicts are resolved.

(2) Knowledge Representation

The need to represent and characterize such factors as the reasoning strategy for knowledge applications and the capability for knowledge enhancement and augmentation, require certain prerequisites:

- a knowledge of the concepts and facts that constitute the problem domain
- an understanding of the domain problems
- skills at solving problems within the domain
- knowledge acquired through experience in the domain

In short, an "expert".

(3) Inexact Reasoning

Some problem-solving applications can result in conclusions that may not be inferrable with certainty. In addition, the database may have omissions or errors. The use of judgmental knowledge must be applied in such circumstances. This capability implies the capacity to: augment operators with measures reflecting a strength or belief in the inferences they embody and the evidence; utilize an inexact inference procedure to make use of the measures; and determine thresholds of acceptability for hypotheses.

(4) User Transparency

The design of the system should be such that it minimizes the level of expertise required for use. This implies that there is a natural language interface, a transparency of the reasoning process, and a method for determining and locating errors in the knowledge base. This will result in a highly interactive user interface module that utilizes the results of all associated and embedded processes in a manner which is invisible to the user.

(5) User Aided Design

It is important to remember that the formation of an initial model of domain knowledge and expertise is at present an empirical process involving extensive exchange between the domain expert and the knowledge engineer. It is essential that the user remain involved throughout the entire design and implementation processes to provide the basic understanding of the overall requirement.

It is equally important for the system designer to remain unimpaired by the existing operational and technical considerations. He must maintain a global perspective and continually ensure that problem solutions are generic when possible, and as universal and transportable as feasible.

SECTION 3
OPERATIONAL SYSTEMS AND STUDIES

The following represent examples of functioning systems and research efforts that relate to the study area. These include:

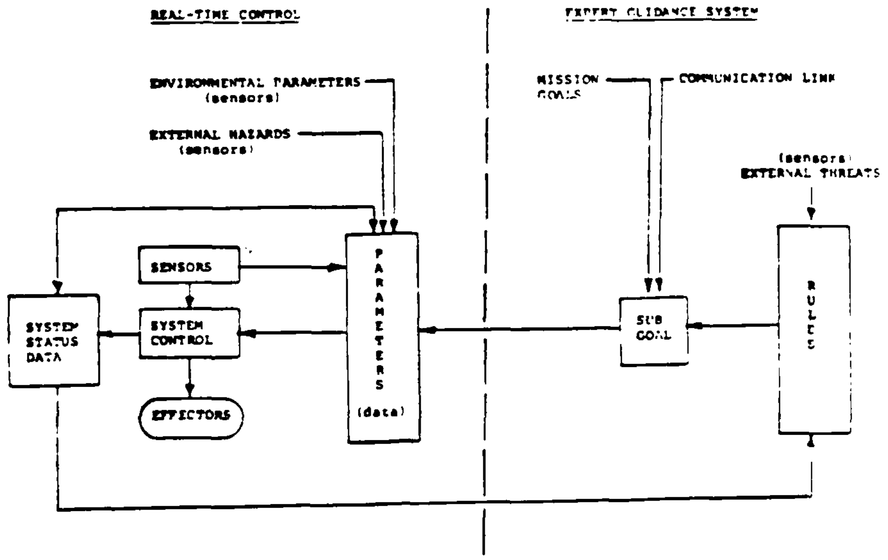
- o Automated Decision Aids - are systems that perform real-time monitoring of internal and external conditions, optimize planned actions, and cope with unexpected events according to the rule-based optimization of a set of alternatives.
- o Expert Systems - are referenced in relation to consultant based systems with optimized resource allocation and inferential data search.
- o Semantic and Frame Architectures - are referenced with regard to relational data base structures and inference data access for textually oriented applications.
- o Generalized Planning Systems - include examples of research efforts and systems that model the planning process.

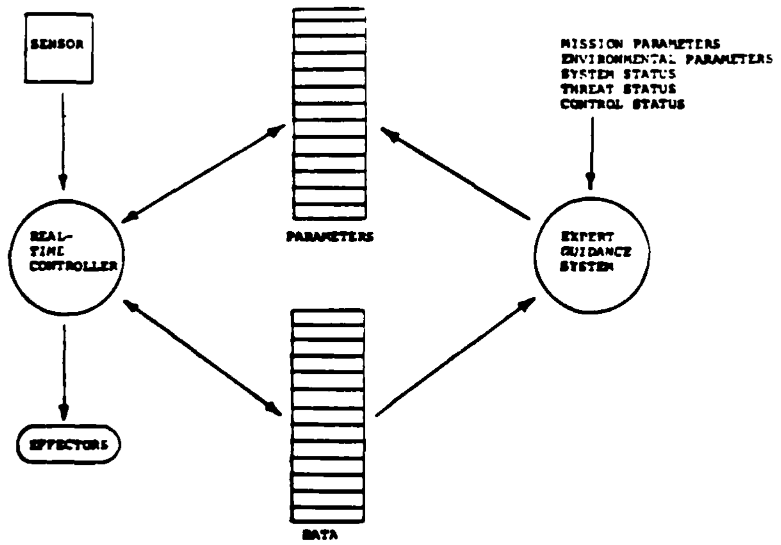
3.1 AUTOMATED DECISION AIDS

Current developmental efforts and existing decision making systems are addressing the need for a capability to evaluate alternative actions to accommodate unexpected events while optimizing the probability of mission success. Moreover, the system must be capable under time-constrained conditions.

There are ongoing efforts at the Marine Systems Engineering Laboratory, University of New Hampshire, and Defense Advanced Research Projects Agency (DARPA) to develop an expert system for control of an autonomous undersea vehicle. The DARPA effort has a Mission Control Logic (MCL) that utilizes a valuated state space, with a partitioned set of operators. The expert system is modeled after a human expert's comprehension of potential unpredicted events and a set of rules representing the judgmental knowledge to be brought to bear on the particular event. As shown in Figure 3-1, the expert system provides real-time monitoring of the environment and situation, as well as time

FIGURE 3-1


REAL-TIME CONTROL
EXPERT GUIDANCE SYSTEM
ENVIRONMENTAL PARAMETERS (sensors)
EXTERNAL HAZARDS (sensors)
MISSION GOALS
COMMUNICATION LINK
(sensors) EXTERNAL THREATS
SENSORS
SYSTEM STATUS DATA
SYSTEM CONTROL
EFFECTORS
PARAMETERS (data)
SUB GOAL
RULES


SENSOR
PARAMETERS
MISSION PARAMETERS
ENVIRONMENTAL PARAMETERS
SYSTEM STATUS
THREAT STATUS
CONTROL STATUS
REAL-TIME CONTROLLER
EXPERT GUIDANCE SYSTEM
EFFECTORS
DATA

constraints. As interruptions to the expected sequence of events occur, the control system dynamically adjusts by revising the mission objective and state sequencing.

The MCL has a simulation implemented on a DEC VAX 11/780 computer which requires about 80 Kbytes of memory and is written in Fortran. The simulation development required three manyears of effort. Similar systems of comparable magnitude utilize Adaptive Maneuvering Logic (AML) at Nellis AFB and NASA/Dryden to perform flight combat simulation.

3.2 EXPERT SYSTEMS

An expert consultant system which has recently been developed under a government research contract is the Interrogator System (see Table 3-1). Similar in nature to BATTLE (a weapon allocation system for the Marine Integrated Fire and Air Support System (MIFASS)). The system utilizes the AI techniques in two ways. First, the effectiveness index of the resources with respect to particular targets is computed and an allocation tree is constructed for determining allocation plans. The effectiveness values of each resource are then used to direct the pruning of the tree and the determination of the optimal allocation.

In addition, Interrogator makes use of personality traits and personnel history to posture a potential enemy threat and courses of action (see Figure 3-2). In this way, the ALBS 2000 gaming module can be linked to test the allocation process.

Interrogator was implemented on a DEC VAX 11/780 and has been downsized to a Convergent Technologies (8086-based) system. It is written in PASCAL and requires about 128 Kbytes of memory.

TABLE 3-1

INTERROGATOR SYSTEM

- Designed for Use on Convergent Technologies (8086-based) microprocessor color workstation

  - inexpensive, available, proven reliable and rugged
  - versatile and easily customized
  - high resolution color image, raster and vector graphics display

- Special Artificial Intelligence Features

  - Brigade S-3 Expert System
  - Wargame module simulating AIR LAND BATTLE 2000 concepts
  - User control over all engagement rules/doctrinal logic/scenario/weapons data
  - Incorporates NBC environment
  - Includes higher levels of command decisions and abstract concepts
  - Personality profiles to construct enemy courses of action

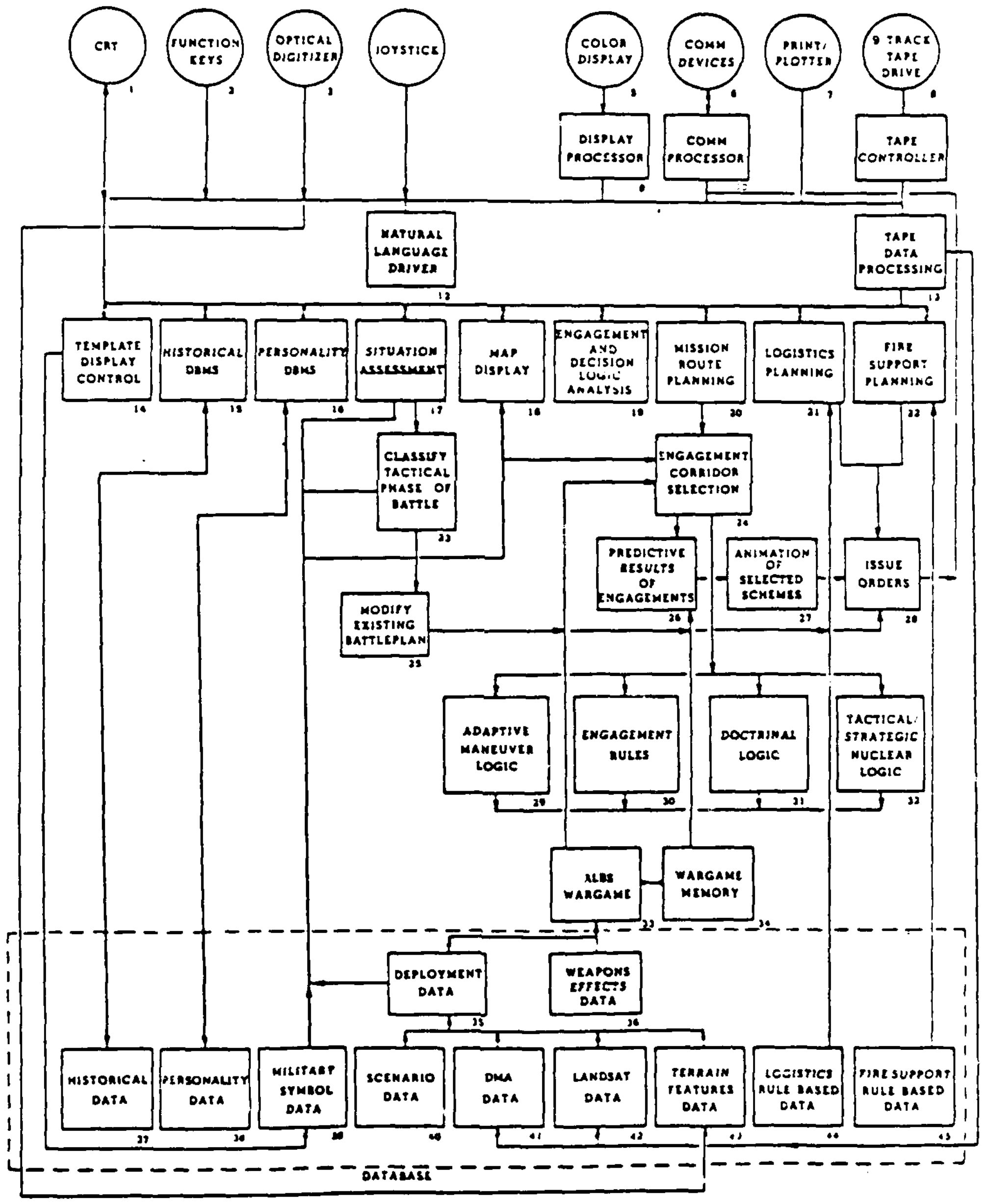
- Unique Display Modes
  - Colored terrain and military symbols
  - Customized mission statement, situation assessment, and maneuver scheme templates
  - Personality and enemy vehicle identification photos.
  - Historical battle plans and dispositions
  - Image generated terrain views (line of sight, masking, etc.) facilitates map reading

- User Friendly Man-Machine Interface Features

  - Easy Menu-Driven Formats
  - Special Function Keys
  - Touch Screen Overlays
  - Natural English Input/Output
  - Windowing Capability

FIGURE 3-2

END PRODUCT SYSTEM FUNCTIONAL BLOCK DIAGRAM



3.3 SEMANTIC NETWORKS AND FRAME ARCHITECTURE

Semantic networks have been used for representation of knowledge since the mid 1960's. Quillian's[1] 1966 semantic memory is considered to be the first semantic network with its roots in Raphael's 1968 SIR[2] and in the work of Reitman.[3]

A semantic network can be defined as a labeled, directed graph in which nodes represent concepts; an arc labeled R going from node n to node m represents that the concept of n is related to the concept of m through the relation R.

The notion of concept is not easily defined, but it can be thought of as anything about which information can be stored and/or transmitted. Various semantic networks have included as concepts prototypical individuals, actions, sets, propositions, facts, beliefs, roles, relations, hypothetical worlds and others.[4]

The basic notions of semantic networks include: nodes to denote objects, concepts, situations; arcs to denote relations (associations) between nodes; and classification arcs to enable hierarchical organizations of knowledge and permit property inheritance.

In general, the reasoning method is a function of the procedures that manipulate the representation and for inference, the principle method is matching through:

- o comparison of network fragments representing data and the knowledge base
- o heuristics to suggest locations to match

Frames generalize the semantic network by providing a common data structure, expectations that allow a frame to match itself to the current situation, and procedural attachments as well as a variety of other inference techniques.

---

[1]Ref. Quillian, W. R., Semantic Information Processing, MIT Press, 1968
[2]Ref. Raphael, D., Semantic Information Processing, MIT Press, 1968
[3]Ref. Reitman, W. R., Cognition and Thought, Wiley, New York, 1965
[4]Ref. Shapiro, S. C., ACM SIGERT Newsletter 63, 1977

In summary, the most important features of semantic nets and frames include:

- an explicit and economical representation of the structure and organization of the domain.
- expectations, defaults, restrictions, and contingencies that direct the reasoning process.
- the procedural attachment that gives frames the event driven capability of rules.

A number of semantic network architectures have been implemented. ZOG is an example of a system developed at Carnegie Mellon University under contract to the Office of Naval Research, DARPA, and the Air Force Avionics Laboratory. It has been implemented on the USS Carl Vincent and has as its objective the provision of an automated library and administrative process.

Another such system is the SNePS semantic network processing system which is a descendent of MENTAL[1]. SNePS is currently implemented in ALISP and runs interactively on the CDC CYBER 173 at the State University of New York at Buffalo.[2]

3.4 PLANNING SYSTEMS

A plan is a hierarchical process that controls the order in a sequence of operations. Among the benefits of a planning procedure are reduced search, goal conflict resolution, and error recovery capabilities. Among the various approaches are nonhierarchical and hierarchical planning, script-based planning, and opportunistic planning.

Nonhierarchical planning conforms to the most commonly understood meaning of planning; namely, a nonhierarchical planner develops a sequence of problem solving actions to achieve each of its goals. It may use problem-reduction or means-ends analysis to reduce the difference between the current state of the world and

[1]Ref. Shapiro 1971, Proceedings of the 2nd International Joint Conference on Artificial Intelligence, 1971

[2]Ref. Shapiro, Associative Networks, Academic Press, 1979

that state which would exist after problem solving. Examples of nonhierarchical planners are STRIPS[1], HACKER[2] and INTERPLAN[3].

The major disadvantage of a hierarchical planner is its inability to distinguish between critical actions and less important details. Thus plans are flawed by interferences between subgoals and corrections are done by testing alternatives for interference avoidance. This results in an expanded search space. In order to achieve a balance between too little and too many details, hierarchical planning was developed. The method consists of sketching a plan that is complete but vague, and refining the vague parts into detailed subplans to result in the detailed solution. Thus, hierarchical planners use a hierarchy of abstraction spaces to develop a plan.

One approach to hierarchical planning is the ABSTRIPS program, an extension of STRIPS. ABSTRIPS determines critical subgoals and ignores others. By ignoring details, one *effectively reduces the number of subgoals to be accomplished* in any given abstraction space.

Hierarchical planning was implemented in its earliest form by Newell and Simon (1972) in their GPS model of theorem proving logic. The GPS approach was slightly different form that of ABSTRIPS. In ABSTRIPS, a hierarchy of abstraction spaces is defined by treating some goals as more important than others, while in GPS there was a single abstraction space defined by treating one representation of the problem as more general than others.

Subsequent implementations of the hierarchical planning approach such as NOAH and MOLGEN[4] are, again, slightly different from either ABSTRIPS OR GPS. ABSTRIPS abstracted critical goals, and GPS abstracted a more general representation of an aspect of its problem space. NOAH abstracts problem-solving operators; it

[1]Ref. Fikes and Nilsson, Artificial Intelligence, N. J., 1971
[2]Ref. Sussman, G. J., A computational model of skill acquisition, American Elsevier, 1975
[3]Ref. Tate, Austin, Treatise, University of Edinburgh, 1975
[4]Ref. Cohen/Feigenbaum, Handbook of Artificial Intelligence, 1982

plans initially with generalized operators that it later refines to problem-solving operators given in its problem space. MOLGEN goes one step further,abstracting both the operators and the objects in its problem space. In all cases, however, hierarchical planning involves defining and planning in one or more abstraction spaces. A plan is first generated in the highest, most abstract space. This constitutes the skeleton onto which details are fleshed out in lower abstraction spaces. Hierarchical planning provides a means of ignoring the details that obscure or complicate a solution to a problem.

A third approach to planning also makes use of skeleton plans but, unlike hierarchical planning, these skeletons are recalled from a store of plans instead of generated. This approach was adopted in one of the MOLGEN systems. The stored plans contain the outlines for solving many different kinds of problems. They range in detail from extremely specific plans for common problems to very general plans for broad classes of problems. The planning process proceeds in two steps: First a skeleton plan is found that is applicable to the given problem and then the abstract steps in the plan are filled in with problem-solving operators from the particular problem context. This instantiation process involves large amounts of domain-specific knowledge, often working through several levels of generality until a problem-solving operator is found to accomplish each skeleton-plan step. If a suitable instantiation is found for each abstracted step, the plan as a whole will be successful.

This approach has much in common with that of Schank[1] and his colleagues. Their approach to natural-language understanding is to use stored scripts (and other, more sophisticated structures) to provide top-down expectations about the course of a story.

[1]Ref. Schank, R.C., Scripts, plans, goals, and understanding, Hillsdale, N. J., 1977

A fourth approach to planning is the Hayes-Roth[1] model which is termed opportunistic and is characterized by greater flexibility than any of the other approaches.

- o Communication is accomplished through the blackboard which contains:
  - initial data or goals
  - hypotheses (partial solutions) at various levels of abstraction
  - support links between different levels
- o Specialists attend to particular areas of the blackboard to:
  - create and modify hypotheses
  - record evidential support between levels
  - ensure consistency of hypotheses
  - focus on promising hypotheses

Specialists are scheduled for activation opportunistically in that there is no particular order. The resulting asynchrony of planning decisions that are made only when required gives rise to the term opportunistic.

Each of the various planning systems must be evaluated for suitability in terms of the unique requirement. That is, depending upon the application, one or more of these systems operating in conjunction may be appropriate. The blackboard approach functionally represents the use of multiple systems if individual specialists were application specific and resulted in the use of multiple planning systems.

3.5 HIGH LEVEL LANGUAGES AND INTERACTIVE PROGRAMMING

The requirement for AI programming to handle knowledge based data structures such as semantic networks as well as provide flexible and dynamic control structures and pattern recognition capability are essential capabilities of an AI

[1]Ref. Hayes-Roth, B., Human planning processes, Rand Corporation, Santa Monica, California, 1982

programming language. LISP, developed by John McCarthy in 1958, has been the basis for much of the AI programming development. The key concepts embodied in LISP include:

- o Symbolic rather than numeric computations.
- o List processing with data represented as linked - list structures.
- o Control structure with aggregation capability to form more complex functions.
- o Recursion as a method describing processes and problems.
- o Representation of LISP programs as linked lists, the same as data.

The predominance of AI program development has been on DEC PDP-1ØS and PDP-2ØS in LISP, principally INTERLISP and MACLISP. The former has extensive user facilities resident in the system while the latter utilizes separate routines and emphasizes speed and efficiency. Other LISP dialects include SAIL, POP-2, QLISP, FUZZY, and PROLOG, with higher level languages like MICRO-PLANNER and CONNIVER implemented in MACLISP. Recently, however, the emphasis has been on the development of hybrid (computational and symbolic) language environments such as SETL and ROSS. In addition, specialized, dedicated LISP machines and relational DBMS systems such as the Britton-Lee will facilitate the user involvement in AI programming environments. These capabilities are fundamental to adaptive software technologies and bode well for the integration and infusion of expert system applications.

SECTION 4

FUNCTIONAL CAPABILITIES REQUIREMENT

Based upon the discussions in Section 3, there are a number of basic functional capabilities that should be considered and evaluated for inclusion in the material/transportation planning system.

The need for a relational DBMS and a LISP-like programming language has been discussed. These capabilities are essential to the use of adaptive software technologies and expert systems. The key is to integrate the operational language, Ada, with the hybrid and knowledge based sub-system languages. The inclusion of their capabilities affords the opportunity to implement any or all of the general system capabilities previously described. These include the knowledge-based systems:

- Automated Decision Aids
- Expert Systems
- Semantic and Frame Architecture
- Generalized Planning Tools

In addition, there are the obvious elements that fall into the class of deterministic simulations and models.

SECTION 5
FIVE-YEAR SYSTEM ELEMENT AVAILABILITY

The rapid advances anticipated in both hardware and software for AI application over the next five years will likely result in a variety of alternatives for implementing the relational DBMS and LISP-like language environments.

In addition to the cost for this "off-the-shelf" hardware and software, there are a number of areas that will require extensive effort if AI techniques are to become an integrated component of the material/transportation planning system. These include:

- The development and implementation of natural language and Ada-based software interface.
- Operation and system analysis to facilitate development of system functional and performance requirements.
- Research and evaluation of knowledge-based systems as applied to this application, including decision aids, semantic network architectures, and expert systems and planning methodologies.
- Development and implementation of knowledge based adaptive software as appropriate for the application.

SECTION 6
DEVELOPMENT PLAN

6.1 *REQUIREMENTS ASSESSMENT*

In view of the vast amount of research and development ongoing in the area of AI applications, it is likely that a number of efforts will provide substantive input in the selection of various hardware and software technologies suitable for this application. As a result, there is an initial requirement for a technology assessment survey.

In addition, there is the obvious need to assess the capabilities and design of the existing WWMCCS facilities.

The purpose of these efforts will be to:

- o Determine baseline functional requirements and capabilities of existing systems.
- o Modify the requirements to provide for reasonable and desirable capabilities that utilize AI techniques and methodologies to enhance overall system performance.
- o Assess the applicability of existing technology and determine candidate areas for research and development.

Subsequent to the above efforts, there will be a review of the candidate hardware and software components with particular emphasis on the assessment of the potential benefits, cost and risk factors. Decisions will be made regarding the prioritization of various components, and a plan for the development and implementation of the objective system will be produced.

6.2 LEVEL OF EFFORT

The activities described above would require initiation two-five years prior to the scheduled installation date. Five years with an increasing level of effort is preferable, but two-three years is possible with intensive efforts. The level of effort if accomplished over the five-year timeframe would range as follows:

Year 1: 2-4 manyears
Year 2: 2-4 manyears
Year 3: 3-5 manyears
Year 4: 5-7 manyears
Year 5: 5-7 manyears

For the two-year timeframe:

Year 1: 10-12 manyears
Year 2: 12-15 manyears

SECTION 7
CONCLUSIONS

The conclusions of this study effort can be summarized as follows:

- o Existing hardware and software technologies will be useful in solving elements of the problem, but not readily adaptable as a total system solution.
- o A requirements and capabilities analysis is recommended to assess the existing facilities and the projected need for enhancements and additions.
- o A technology assessment survey is recommended to evaluate the potential utility for AI applications and methodologies in the material/transportation planning applications.
- o It is expected that substantial software development will be required to supplement the selected AI technique.· This is particularly true for the requirement to interface with Ada as the base language.

Technology Transfer to the Conventional Force Deployment Problem

Edison Tse
Department of Engineering-Economic Systems
Stanford University
Stanford, CA

Technology Transfer to the
Conventional Force Deployment Problem

There are two Automatic Data Processing systems that resemble the one perceived to be developed for applications in Conventional Force Deployment (e.g., (1) in force planning and replanning, (2) logistics support planning and replacing and (3) retrospective analysis of operational data). In these few pages, we shall briefly describe these systems, their status and the costs for their development. We shall also briefly comment on transferring their technology to the conventional force deployment problem. They are the KNOBS/TEMPLAR for the Tactical Air Control Center (TACC) and the Mission Planning System for SAC.

KNOBS is an applied research system developed by MITRE for TACC. Its goal is to support the design of individual strike missions within the air tasking process. It operates by assisting the user as he completes the various portions of predefined mission forms which identify the aircraft, ordnance, communication frequencies, attack times and other factors that specify a mission against a particular target (an enemy airfield).

For example, if a flight of F-111s is going to be used against a SAM defended target, KNOBS will tell the planner if the F111 does not have the required range, if it cannot get there in the specified time, and if its ordnance should include ECM. This kind of consistency management is the primary goal of KNOBS.

KNOBS also provides planning support and autonomous planning facilities. The system is able to suggest alternatives for the resources used in missions that meet all of the known requirements, it can rank order those candidates by their appropriateness, and it can complete an entire mission design if it is given a few critical components. To suggest alternatives, KNOBS essentially tries all possibilities and rules out the ones which are unreasonable; to plan autonomously the system checks all candidate resources (in a specified order) until it finds the first set that satisfies all constraints.

From a technical standpoint, KNOBS is a research prototype (as opposed to an application system) that has focussed on the development of AI technologies in response to the requirements of the mission planning domain. It demonstrates a way in which constraints, heuristic rules, inheritance, and automatic deduction mechanisms can be orchestrated together into a single system that effectively produces valid mission plans.

KNOBS provides a substantial natural language component which creates a uniform interface to the bulk of the system's facilities. Modifiability is provided by allowing the user to input new rules and constraints in English in real time during a planning session. The system explains its behavior by monitoring the progress of its deduction mechanisms, and then translating a trace of their actions into English. In addition, the Natural Language component allows the user to query the system's knowledge base, and to make new commitments in the process of completing a mission. To a limited extent, KNOBS is able to carry on a dialogue with the user as he performs these functions.

TEMPLAR (Tactical Expert Mission Planner) is an advanced developmant model prototye of KNOBS. It is now under contract development by Advanced Information and Decision Systems. In this development, in addition to engineering the capabilities already present in KNOBS for Offensive Counter Air (OCA), new capabilities that are necessary to provide a useful tool for a spectrum of tactical air mission planning problems (e.g., interdiction, close air support, defensive counter air) will be added. Moreover, a better man-machine interface will be developed in TEMPLAR. Thus one can view TEMPLAR as a successor to KNOBS, which is moving closer to an operational system.

KNOBS has developed over a period of 5 years at a cost of approximately 2.5 million. TEMPLAR is estimated to cost around 1.5 million, and it is estimated that to bring TEMPLAR into operation, an additional 3 to 4 million would be required.

The Strategic Mission Planning System for SAC was developed by Systems Control, Technology. The system is based on dynamic programming algorithm, coupled with path optimization, which allows one to determine the optimal flight path, target and weapons assignment. The joint optimization can be applied to the case of a single bomber, or cruise missile, and the resulting flight path and weapons allocations can be incorporated into the SIOP.

While the basic methodology is based on OR optimization methods, when the full many weapons to many targets problem is addressed to the system, it becomes too large to handle vigorously; heuristic methods are then introduced to deal with the path optimization problem. The Strategic Planning System has a good graphic and a front-end user interface which allows the system to be used interactively. The system is now being developed for an operational phase. The total development cost from design through initial operating capability is approximately 7 million, with approximate breakdown given in table 1.

The basic distinction between the two systems described is that while Strategic Mission Planning is built on a simple structure and then is extended to more complex and realistic situations (bottom up),

Table 1: Approximate Breakdown of Total Development Cost

systems / phases	KNOBS/TEMPLAR	Mission Planning
Design	KNOBS ± 2.5 million	± 400 K
Feasibility		± 800 K
initial demo/ test bed	TEMPLAR ± 1.5 million	± 2.4 M
Operational	(Estimate) 3 - 4 M	± 3.4 M
Total	±7.5 million	± 7 million

KNOBS/TEMPLAR is started with the fact that the problem to be dealt with is too complex to be handled completely analytically, and thus uses artificial intelligence (AI) as its base to integrate diverse knowledge to assist in the planning process (top down). For the conventional force deployment problem, it seems that an approach, which starts from realizing complexity while incorporating and exploiting the existing OR models that had been built to represent our knowledge source would be a promising approach to the problem.

ORI
Silver Spring, Maryland 20910

TECHNICAL AREA REPORT: MATERIAL/TRANSPORTATION PLANNING MODELS

BY: S. A. KLEIN

15 SEPTEMBER 1983

PREPARED FOR:
SOFTWARE ARCHITECTS AND ENGINEERS, INC.
1401 WILSON BOULEVARD
ARLINGTON, VA 22209

ACKNOWLEDGEMENT

The author wishes to acknowledge the contributions of F. Hopkins, G. Hamilton, B. Buc, and G. Holiday in the preparation of this report.

PRECEDING PAGE BLANK-NOT FILMED

TABLE OF CONTENTS

1. OVERVIEW

1.1 INTRODUCTION

The purpose of this report is to support an effort by the Institute for Defense Analysis that is developing estimates of the cost and scope of component subsystems for the next generation World Wide Military Command Control System (WWMCCS). This report addresses the technical area of material/transportation planning, which is defined to include "(1) force planning and replanning (2) logistics support planning and replanning (3) retrospective analysis of operational data." The objective of this effort is to "revisit the area of operations research models and heuristic planning systems to see if some of that technology can be applied to improve military planning."

The report is organized as follows. Section 1.2 presents the study conclusions. Section 2 discusses the functions involved in military material/transportation planning, identifies the classes of models/tools that may be applicable to support these functions, and outlines some issues that must be considered in system implementation. Section 3 presents some examples of functioning systems related to the study area, both military and commercial/industrial. Section 4 describes some examples of subsystems and building blocks that may be useful in system development. Section 5 discusses some anticipated types of system elements and their future availability. Section 6 presents a recommendation for the effort required to prepare a detailed development plan.

1.2 CONCLUSIONS

The conclusions of the study are as follows:

1. While some commercial off-the-shelf hardware/software may be useful, the pursuit of commercial sector applications with a view toward adapting them is not likely to be a fruitful approach to development of the desired system.

2. Extensive development effort is likely to be needed with specific orientation toward the intended application. Much of this effort is probably occurring already under the sponsorship of a variety of DOD-related organizations. However, these activities may need to be supplemented or redirected.

3. An initial exploratory requirements analysis (outlined in Section 6) is recommended. The effort is focused on obtaining user views and surveying ongoing related research/development activities.

2. FUNCTIONAL REQUIREMENTS

This section presents an overview of the functional requirements for a computer system to support military material/transportation planning. Section 2.1 identifies the areas in which this activity can be supported by computation. Section 2.2 discusses the classes of models/tools that may be useful in providing computational support and identifies the areas in which each model/tool is potentially applicable. Section 2.3 presents some issues that are critical in implementing a system of this type.

2.1 FACTORS IN MATERIAL/TRANSPORTATION PLANNING

This section describes the scope of the military material/transportation planning problem and identifies the areas in which computational support may be useful.

Table 1 lists some of the factors that must be addressed in material/transportation planning for military operations. The basic driving factors are force configuration, mission, and geography. These factors result in a set of support needs and related conditions (e.g., priorities) for the mission-performing forces. Additional needs (and associated conditions) must be included to support the supporting forces. The resulting overall support needs become the transportation system workload.

The performance of the transportation system depends on the available resources and facilities, and on the constraints imposed by geography, geopolitics, and the military situation.

TABLE 1

FACTORS IN MATERIAL/TRANSPORTATION PLANNING

- Force Configuration and Mission
- Identification of Support Needs
  - Consumables
    - Groceries
    - Maintenance Items
    - Water
  - Ordnance
  - P-O-L
  - Spares
  - Special Items
- Support Conditions/Constraints
  - Initial and Continuing Quantities
  - Allowable Delay Time
  - Transportability
  - Priority
  - Criticality (Loss Impact)
- Resources and Facilities
  - Personnel
  - Transport Resources
    - How Many
    - Where Are They
    - What to do About Attrition
    - Capability and Capacity of Each Type
  - Facilities
    - Refueling/Transshipment
    - Unloading
    - Warehousing/Storage
    - Coordination
  - Intertheater vs Intratheater
- Geography and Geopolitics

Table 2 shows the areas of consideration in material/transportation planning supportable by computation.

2.2 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS

Table 3 identifies classes of models/tools potentially applicable to the areas of military material/transportation planning supportable by computation. The table briefly describes each model/tool and identifies its areas of applicability.

2.3 IMPLEMENTATION ISSUES

In implementing a system using the models/tools listed above, there are some issues that must be considered. These issues represent common pitfalls to be avoided or characteristics of successful model/tool applications. These issues include:

1. Availability of Data. About five years ago, an official in charge of development of a major military C^3 system stated that the biggest problem encountered in using the system was the lack of data. It turned out that a high percentage of the data required for the operation of the system (and assumed to exist by its designers) was not, in fact, being collected.

   The algorithm classes identified for potential application have varying degrees of sensitivity to the quality of the input data. If poor quality data are available, it is preferable to choose a less sensitive algorithm. In general, the exact optimization methods are most sensitive and the statistical and probability-based models are least sensitive.

TABLE 2

CONSIDERATIONS SUPPORTABLE BY COMPUTATION

Calculation of Support Needs (including needs of the support system itself)

Packing Ships, Planes, etc.

Facility Operations Performance (harbors, airports, warehouses, etc.)

Routing and Flow
- Intertheater
- Intratheater

Effects of Attrition and Reliability

Integration of the Planning Process

Data Management
- Data Collection, Storage, and Query
  - Raw Data
  - Previous Cases and Results
- Development/Update of Modeling Relationships

TABLE 3 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS

MODEL/TOOL CLASS	DESCRIPTION	APPLICABILITY/REMARKS
APPLICATION-SPECIFIC MODELS	Models may include:	o Applicability: - Calculation of support needs - Facility performance - Planning process integration
	o Independently developed relationships for requirements, costs, performance, and other factors	
	o Bookkeeping relationships that combine effects due to individual system elements	o There is an overlap between this class of model and deterministic simulation models.
	o *Embedded optimization models*	o *Microprocessor spread*-sheet calculator programs were originally developed to support relatively simple cases of this class of model.
	o Data/results from other models	
		o For an example, see the FAST model (Section 3.4)

TABLE 3 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS (CONTINUED)

MODEL/TOOL CLASS	DESCRIPTION	APPLICABILITY/REMARKS
INPUT-OUTPUT MODELS (also called Leontief models)	Solves for required economic inputs (goods and services) given desired final economic output and unit input requirements. Widely used in macroeconomics. Linearity usually assumed to simplify computation.	o Useful for resolving support requirements including the support system itself. The NAVCOMMSTA Resource *Allocation Model* (see Section 4.1) is essentially a nonlinear input-output model.
OPTIMIZATION METHODS		
- EXACT	These are the traditional mathematical programming models -- *linear*, *non-linear*, *integer*, dynamic, etc.	o Applicable to all areas provided there is agreement between the application and the detailed model conditions/assump*tions*. Solution methods exploit the specific mathematical structure assumed for each model.
- HEURISTIC	These are methods that are frequently used when exact optimization models are too *cumbersome*, costly, or difficult to apply. Heuristics are often partial or non-rigorously modified versions of exact methods. Solutions are generally local minima; globally optimal solutions are not guaranteed.	o Same as for exact methods

TABLE 3 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS
(CONTINUED)

MODEL/TOOL CLASS	DESCRIPTION	APPLICABILITY/REMARKS
- STATISTICAL	This is a class of methods that use statistical techniques for estimating the values of objective functions in optimization problems without actually solving the problems	o Possibly useful in the initial phases of planning studies as surrogates for exact or heuristic methods (that are intended to be used later). o Also possibly useful for determining need for further effort using an exact or heuristic method.
ANALYTIC PROBABILITY	These are the traditional queuing, inventory, reliability, and other applied probability models. Relationships are derived by analysis and numerically evaluated by computer.	o Primarily applicable to: - Effects of attrition and reliability - Delay issues in routing/flow and facility operations performance - Calculation of support needs, where the demands are probabilistic

TABLE 3 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS (CONTINUED)

MODEL/TOOL CLASS	DESCRIPTION	APPLICABILITY/REMARKS
NETWORK MODELS		
- DETERMINISTIC	These are exact and heuristic optimization methods tailored for application to network problems. It is sometimes possible to recast non-network problems in a network form.	o Applicable to routing and flow area, with particular focus on optimizing the flow of supplies in the system, scheduling vehicles, etc.
- PROBABILISTIC	These are analytical probability models tailored for application to networks.	o Applicable to routing and flow area with particular focus on estimating delays and end-to-end transit times
SIMULATION		
- DETERMINISTIC	These are essentially application-specific models in which each execution of a model evaluates a single instance of the operation of a given facility or system for a specified period of time. The relationships in the model are	o Primarily applicable to facility operations performance. May be applicable to routing/flow and vehicle packing. E-LOTS model (see Section 4.2) is an example.

TABLE 3 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS
(CONTINUED)

MODEL/TOOL CLASS	DESCRIPTION	APPLICABILITY/REMARKS
	primarily related to the detailed decisions and performance factors involved in facility/system operation.	
- PROBABILISTIC	These are similar to the deterministic simulations, except pseudo-random processes are used to select the outcomes of decisions and chance events and the duration of inter-event periods where these are governed by probabilistic phenomena.	o Applicability: - Facility operations performance - Any application where an analytic probability model is too difficult computationally. o May require numerous runs because of effects of randomness.
COMPUTER GRAPHICS		
- THREE-DIMENSIONAL	These are the type of graphics system used in computer-aided design of automobiles, buildings, and other structures.	o Applicable to vehicle packing problem.

TABLE 3 CLASSES OF POTENTIALLY APPLICABLE MODELS/TOOLS
(CONTINUED)

MODEL/TOOL CLASS	DESCRIPTION	APPLICABILITY/REMARKS
- MAPPING	These are the type of graphics system used for display of geographic data.	o Applicable to: - Routing and flow - Planning process integration
REGRESSION ANALYSIS	These are usually components of statistical packages such as SAS, SPSS, and BMDP.	o Applicable to development/update of modeling relationships and analysis of probabilistic simulation results
DATA BASE MANAGEMENT SYSTEMS	Provide capabilities for data collection, storage, and query	o Applicable to data management area
EXPERT SYSTEMS	Provide capabilities for programming the heuristic decision rules and diagnostic methods of an expert and implementing an interface to allow an inexpert user to interact/dialogue with these rules/methods.	o Applicable to planning process integration.

2. Formulation of an Objective Function. In most commercial/industrial applications of optimization models, the objective function is clearly related to profit maximization, the overall goal of the using organization. In military material/transportation planning, the objective function may not be so easily formulated. In a rapid response situation, the military commander wants to get the job done; he is probably not interested in minimizing his costs or maximizing some measure of efficiency. Since the relationship between the mathematical objective function and the organizational objective is less clear in the military case, it is likely to become a matter of controversy. This can present a serious obstacle to the system development process.

3. Sensitivity to Underlying Assumptions. In a manner similar to their sensitivity to input data quality, the model/tool classes have varying degrees of sensitivity to their underlying assumptions. The exact optimization methods tend to be most sensitive, because they exploit the detailed mathematical structure of their models to produce solutions.

4. Level of Expertise Required to Operate Model/Tool. Some of the model/tool classes require extensive expertise to operate the model/tool and interpret its results. This is particularly true of the probabilistic simulation models, which require statistical analysis of several runs to ensure validity of the results. If the system is intended for use by military command/staff personnel, it is preferable to design a system architecture that minimizes the model/tool expertise required of the user. This suggests an architecture in which the command/staff personnel interact with a relatively highly aggregated application-specific model that incorporates data/results from other models (that require more expertise to operate) and/or has the other models embedded in a manner that makes their complexity transparent to the user.

5. Choice of Models/Tools. It is best to let the selection of a model/tool result from a detailed study of the application. Immersion of the analyst in the application is an essential prerequisite to this selection. Deciding on a model/tool before the application is fully understood usually has disappointing results. However, it is also important for the analyst to avoid becoming bound by existing operational considerations.

Frequently, the problem posed to the analyst turns out to be a symptom of a much broader problem. If a solution to the broader problem is both technically and politically feasible, it may be preferable to allow the analyst to focus on solving it.

6. User Involvement. It is very important to involve the users in the design of the system. This does not mean that the users should define the system -- they will generally not know what is technically feasible and potentially beneficial. However, by having some form of user involvement, the success of the implementation effort is significantly enhanced.

3. EXAMPLES OF FUNCTIONING SYSTEMS

This section presents examples of functioning systems related to the study area. The examples include both military and commercial/industrial systems. The examples and their relationships to the system of interest are as follows:

- <u>Electric utility control systems</u> - are the computer - communications systems that perform real-time economic optimization and security assessment in electric power systems. These control systems are probably the example most analogous to the system of interest because of:
  - The scope of the application
  - The classes of functions/algorithms and real-time performance requirements
  - The size and scope of the research community contributing to application development

- <u>Commercial trucking and shipping company systems</u> - that are published examples of business applications of the relevant methodologies.

- <u>The Force Assessment Deployment Simulation</u> - A system that is being used to train some of the military commanders who will eventually use the system being addressed in this paper.

- A set of relevant models in the aviation area - that are currently being used for various purposes.

3.1 ELECTRIC UTILITY CONTROL CENTERS

An electric utility control center is a facility that exercises real-time control over the generation and transmission (and, in some cases, the distribution) operations of an individual utility or a group of utilities (power pool). The two principal functions of a control center are:

A. Economic optimization of the utility operations, and

B. Assessment of the security of the power system against the effects of potential failures that could cascade and result in blackouts.

To perform both of these functions, control center personnel interact with algorithms that are specialized versions of mathematical programming techniques. Data to feed these algorithms is collected from remote monitoring equipment located in substations and generating plants. There are over 150 control centers (existing or planned) around the world. Each center collects data from up to 300 remote terminal units, each of which may monitor a number of algorithm-related parameters.

The economic optimization of a utility takes place over a hierarchy of planning/implementation horizons that range from 20 years to 6 seconds. Table 4 shows the hierarchy of economic optimization functions. These functions address both the generation of power by the utility for its own customers and the interchange (purchase or sale) of power with other utilities. This power interchange may occur under a long-term planned or a short-term ad hoc arrangement. Included with the economic optimization functions are algorithms that support utility personnel in the real-time assessment of short-term interchange proposals.

TABLE 4

ELECTRIC UTILITY
ECONOMIC OPTIMIZATION HIERARCHY

OPTIMIZATION FUNCTION	TIME FRAME	DESCRIPTION
Facility Planning	10-20 Years	Given a long-range demand forecast, determine the facilities that must be constructed to ensure adequacy of generation capacity and reserves.
Maintenance Scheduling Hydro Scheduling	1 Year	Prepare schedules of maintenance operations and hydroelectric generation that provide adequate reserve capacity and meet constraints on stream flows.
Production Scheduling	Several Weeks	Schedule type of generation considering anticipated demand, relative costs, and status of hydro reservoirs and fuel stockpiles.
Unit Committment	8 Hours to 1 Week	Schedule start-up/shutdown of generator units considering availability of units, relative costs, demand profile, and spinning reserve requirements.
Economic Dispatch	10-30 Minutes	Given the set of generators currently spinning, determine how the generator loadings should be adjusted to accommodate changes in system load.
Generation Control	1-10 Seconds	Adjust the generator loadings to meet the system load in accoraance with the economic dispatch results.

The security assessment algorithms are generally of a larger computational scale than the economic optimization functions. Security assessment algorithms are frequently formulated as a nonlinear programming problem where the objective function is a minimization of an imbalance or estimation error and the constraints are a version of the alternating current electric network equations of the power system. Typical networks have hundreds of nodes.

To perform these functions, a typical control center will have a dual or quad configuration of a superminicomputer. For example, a popular machine in recent system has been the Gould/SEL 32, reputed to be the fastest minicomputer on the market. Utilities have also shown interest in the Floating Point systems AP-120B Array Processor.

Development of the algorithms is an ongoing process that has involved extensive support by individual utilities, control center hardware/software vendors, universities, government agencies, and government or privately sponsored research institutes around the world. The combined budgets for research on these algorithms of the two major U.S. organizations supporting this research totals about $10-15 million per year. This does not include private research by vendors and individual utilities, or support in other countries, many of which have extensive research programs in this area.

3.2 TRUCKING INDUSTRY APPLICATION

Barker, Sharon, and Sen (Reference 1) describe a set of models they have developed to improve profitability at ANR Freight System, which owns a number of trucking companies.

The model development effort was initiated because of a severe capacity problem at a major break-bulk terminal. Management established a project team with the objective of developing a tool that would assist in the linehaul planning process. The effort resulted in two models, a Freight Flow Model for analyzing the "less-than-truck-load" (LTL) operations, and a Minimum Revenue Model for analyzing the "truck-load" (TL) operations. The models use a combination of mathematical programming and application-specific methods. Figures 1 and 2 show schematic diagrams of the models.

The development of these models has resulted in increased profits of $9 million per year to ANR. The effort was the winner of the 1981 Management Science Achievement Award offered by the College on the Practice of Management Science of the Institute for Management Sciences.

3.3 BETHLEHEM STEEL MARINE OPERATIONS PLANNING AND SCHEDULING SYSTEM (MOPASS)

The Bethlehem Steel MOPASS model, described in Reference 2, was developed to provide a capability for both annual planning and short-term spot decisionmaking. The model, illustrated in Figure 3, includes:

- o A voyage cost estimating module (VOYEST)
- o An overall fleet operations planning module (PREEMPT) that uses an embedded linear programming optimization module
- o Single and multiple vessel scheduling modules
- o User-oriented information files, management and operations reports.

The fleet being managed is oriented toward carrying various bulk materials (e.g. ores) required for Bethlehem's steelmaking operations. Vessels are also chartered by Bethlehem to and from outside parties. The system has been in routine daily use for over four years.

3.4 FORCE ASSESSMENT DEPLOYMENT SIMULATION (FAST)

The FAST model is both an application-specific model and a deterministic simulation. It supports the top-level, integrated planning function for strategic deployment operations. The model incorporates data/results derived from more detailed studies and models such as E-LOTS

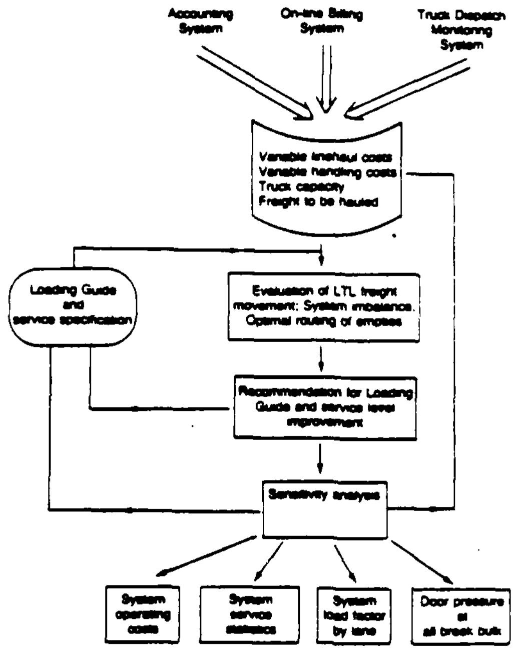

Accounting System
On-line Billing System
Truck Dispatch Monitoring System
Variable linehaul costs
Variable handling costs
Truck capacity
Freight to be hauled
Loading Guide and service specification
Evaluation of LTL freight movement: System imbalance. Optimal routing of empties
Recommendation for Loading Guide and service level improvement
Sensitivity analysis
System operating costs
System service statistics
System load factor by lane
Door pressure at all break bulk

FIGURE 1. ANR FREIGHT LINES FREIGHT FLOW MODEL SCHEMATIC (from Reference 1)

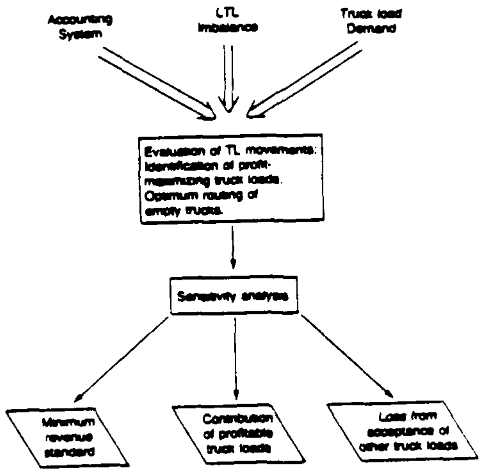

Accounting System
LTL Imbalance
Truck load Demand
Evaluation of TL movements: Identification of profit-maximizing truck loads. Optimum routing of empty trucks.
Sensitivity analysis
Minimum revenue standard
Contribution of profitable truck loads
Loss from acceptance of other truck loads

FIGURE 2. ANR FREIGHT LINES MINIMUM REVENUE MODEL SCHEMATIC (from Reference 1)

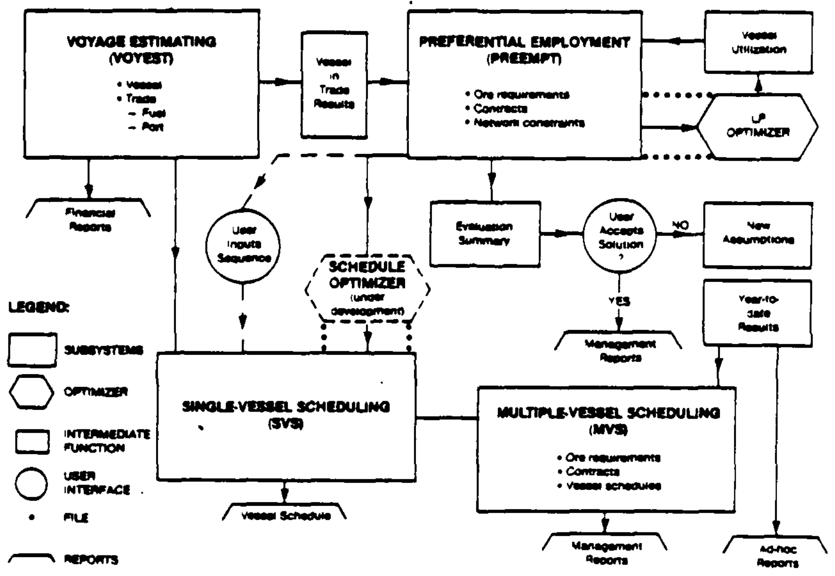


FIGURE 3. BETHLEHEM STEEL MARINE OPERATIONS PLANNING AND SCHEDULING SYSTEM (From Reference 2)

(discussed in Section 4.2). FAST is an outgrowth of an effort at ORI to develop a microcomputer-based model for use at the U.S. Army War College to provide senior-level commanders an understanding of strategic deployment operations and limitations. Table 4 describes the model features.

3.5 AVIATION-RELATED MODELS

This section describes four different hierarchical levels of models used in the commercial, Federal non-defense and defense sectors that can be adapted for DoD use and linked to provide estimates of the following factors:

- element configuration to calculate aviation fuel use,
- fuel requirements by operational characteristics,
- impact of alternative mission specifications within a large scale operation upon fuel requirements, and
- impact of alternative technological developments or procurement procedures upon fuel requirements.

A list of these models and their applications is presented in Table 5. The lowest level of models from the concept of systems optimization are the element models. These models perform calculations on component performance and would provide input into the next level or mission models. The goal of the mission models is to estimate fuel use by individual flights. These models can optimize fuel use by specifying optimal flight paths. The optimal flight path information is used as input into the tactical models that analyze operations that are combinations of individual missions. The long range models are designed to plan the equipment acquisitions that would optimize the achievement of the AF goals. Thus, these models require inputs on energy use from alternative tactical configurations.

TABLE 4
FORCE ASSESSMENT DEPLOYMENT SIMULATION (FAST) FEATURES

- o Designed for Use on Osborne Executive* Portable Computer

  low cost
  accessible
  inter-active

- o User Oriented Data Files

  100+ force components (variable size, supply, lift reqts)
  7+ ship types (variable speed, capacity, load/unload time)
  12 ship inventory changes over 60 days (for each type)
  2 ship POEs (east and west coast)
  10+ aircraft types (variable cargos, capacity)
  5+ aircraft inventory changes (for each type)

- o User Controlled Scenario Factors

  Simulation time (90 day max.)
  Change in combat intensity (influences resupply reqts)
  Availability of Suez Canal (changes distances when closed)
  Theater stockage reqts (level and arrival time)
  Variable sea lane distances (for other than S.W. Asia)

- o Extensive Inter-Active Features

  Screen oriented instructions
  Direct input of unit deployments by air or sea
  Rapid changes to force deployment lists

- o Sensitive to Cargo Pipeline Capacities

  Air cargo delays vs. ALCE unit arrival
  Sea cargo delays vs. COSCOM arrival
  Air traffic limited to 4200 tons/day
  Air and sea port capacity limits
  Direct access to all input data files

*Registered trademark of Osborne Computer Corporation.
The Osborne version is an ORI proprietary system.

TABLE 5

ILLUSTRATIVE AVIATION ENERGY MODELS

MODEL TYPE	VEHICLE TYPE	ORGANIZATION	MODEL CLASS
LONG TERM			
NASA Aeronautical Project Evaluation System	Aircraft	Office of Aeronautics and Space Technology, NASA, ORI	Simulation
MOPASS	Vessels	Bethlehem Steel	Optimization
TACTICAL			
NATO TACAIR	Aircraft	ORI, DOD	Simulation
MOPASS	Vessels	Bethlehem Steel	Simulation
MISSION			
Eastern Airlines	Aircraft	Eastern Airlines	Simulation
Pratt & Whitney	Aircraft	Pratt & Whitney	Simulation
FAA	Aircraft		
SIRO	Helicopter	Army	Simulation
ELEMENT			
Knapsack			Optimization
KONFIG	Aircraft	NASA/Lewis	Simulation

3.5.1 Element Models.

The optimization of energy use on an individual aircraft can be divided into two categories: aircraft performance and aircraft utilization. Aircraft performance is determined by the engineering characteristics of the aircraft. The KONFIG model analyzes the energy consumption of alternatively configured gas turbine aircraft engines. Aircraft utilization can be improved by increasing the load carried by the aircraft. The Knapsack problem is an optimization algorithm that maximizes the number of different sized containers in a storage space. It could be useful for configuring storage in transport aircraft.

3.5.2 Mission Models.

The mission models are useful for examining the impact of alternative flight paths upon fuel use. These types of models are used by Pratt and Whitney to formulate aircraft/engine designs, and Eastern Airlines to maximize revenue on their route structure. The FAA has installed a mission model on an Apple microcomputer, with the goal of developing a system that can be used in real-time in an aircraft.

The Army Aviation Research and Development Command uses the SIRO family of computer models to estimate mission helicopter performance, including energy use. The program is composed of two parts. The first estimates energy use for vertical flight, while the second estimates energy use for horizontal flight.

3.5.3 Tactical Models

Tactical models are very useful for examining energy consumption in peacetime under alternative simulated combat situations. They provide useful inputs in the decision making process on locations of fuel supply depots and optimization of the levels of fuel inventory.

The TACAIR model characteristics are illustrated in Figure 4. Tactical aircraft play a crucial role in the first 30 days of a NATO/Warsaw Pact Center Region war. With a fixed NATO tactical aircraft force level, increasing the number of sorties flown per day by each aircraft may influence the air superiority and air-to-ground TACAIR campaigns. The cumulative air-to-ground sorties, in turn, directly affect the land combat campaign. This effort examines various planning, programming, and policy options to increase the planned sortie rates and the impact of these increased sortie rates on the total achieved number of air-to-ground sorties flown. The cumulative number of air-to-ground sorties is then used to estimate the number of equivalent armored divisions these sorties can potentially destroy.

3.5.4 Long Range Models.

Long-run models are used to analyze the development, acquisition or retirement of equipment used in the aviation or vehicle fleet. The MOPASS model, described in Section 3.3, is used by Bethlehem Steel to optimally plan vessel deployment and requirements for several years in the future. A risk analysis program has been developed to assist in determining the variation in expected return and risk associated with varying the number of vessels in the fleet. Risk analysis should be incorporated in energy/aviation models.

The NASA Aeronautical Project Evaluation System (NAPES) was developed to assist NASA in evaluating the benefits and cost of NASA's technology development plans. The methodological structure of NAPES is presented in the variable precedence diagram in Figure 5. The variables on the left hand side of the diagram are exogenous variables, that are used as inputs in the initial time period of the simulation. Many of these variables become endogenous

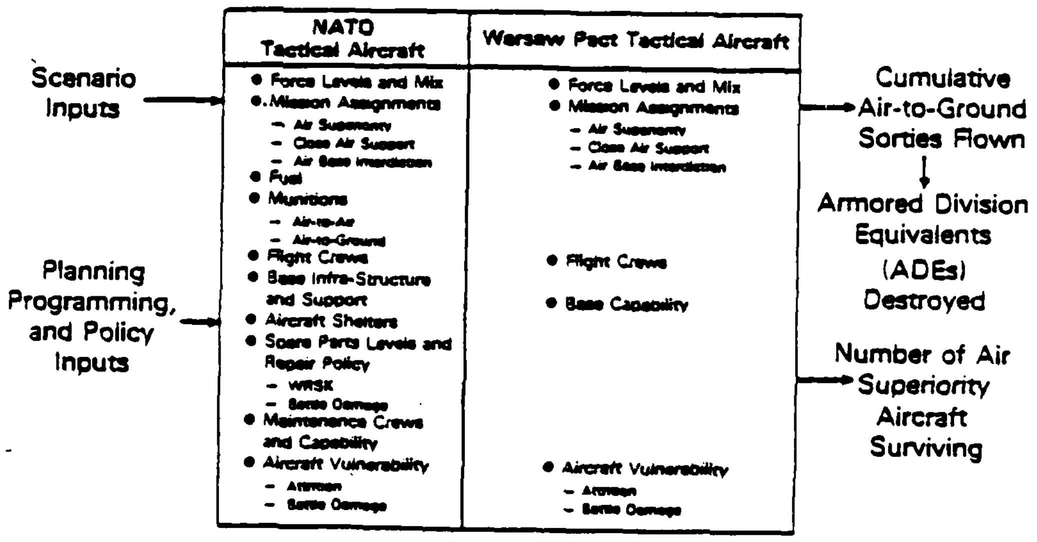


FIGURE 4. OVERVIEW OF TACAIR MODEL USED FOR OSD PA&E

after the initial year and are computed internally. The lines connecting the variables show the relationship of each variable to others in the algorithms used within the system, i.e., the precedence order of each variable in the calculations. There are six logical sections of NAPES, indicated in Figure 5: investment requirements, fleet composition, environmental, travel demand, fuel consumption and airline operations.

The model is capable of performing sensitivity analysis on an extensive set of variables to determine the importance of the analyses to variations in the key independent variables of the study, including economic growth, passenger load factors and fuel prices. Sensitivity analysis is another technique for evaluating the level of risk in a base case solution.

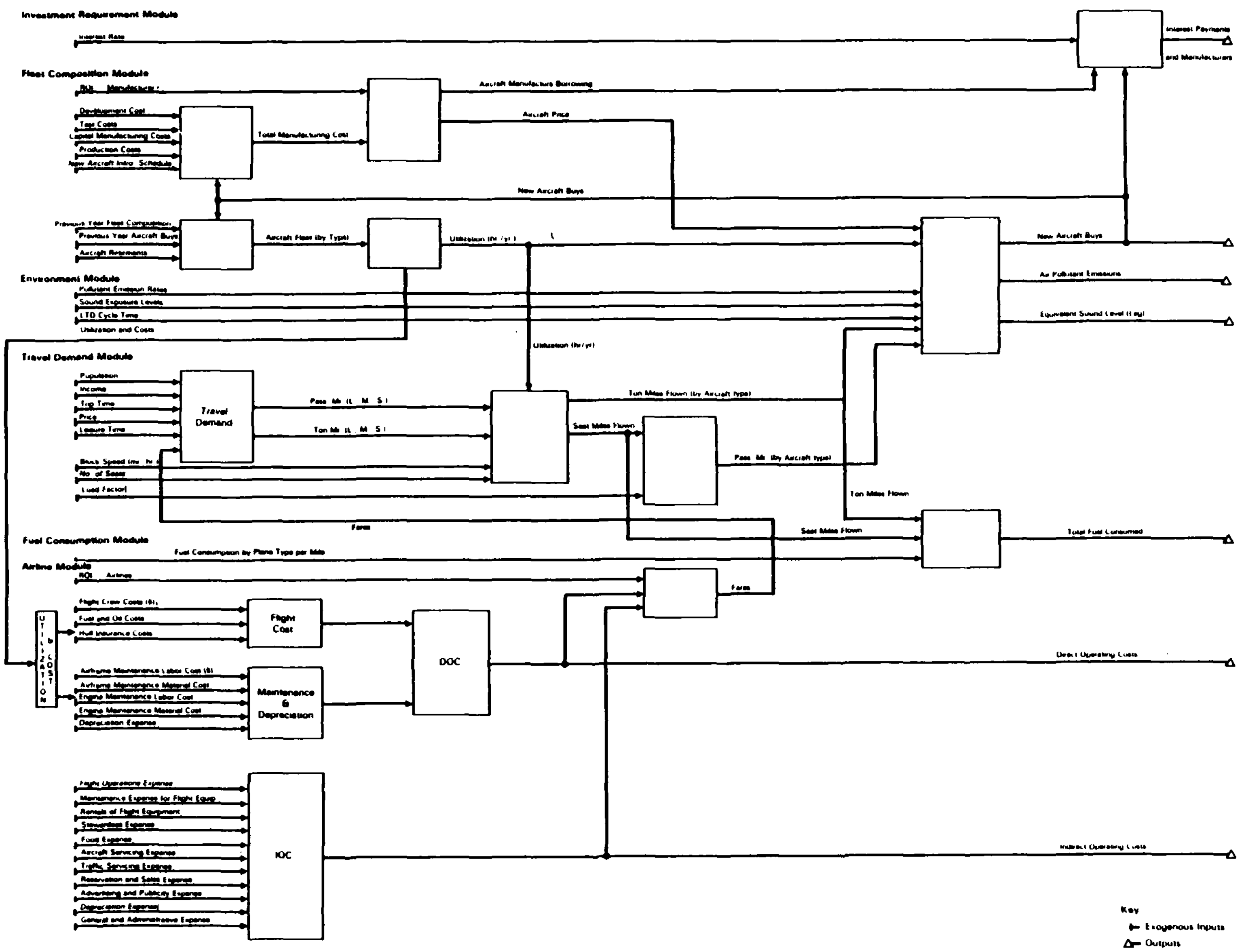


FIGURE 5. NASA AERONAUTICAL PROJECT EVALUATION SYSTEM

4. EXAMPLES OF FUNCTIONAL SUBSYSTEMS

This section provides some selected examples of functional subsystems that may be used as functional or conceptual building blocks for an overall military material/transportation planning system. The examples include:

- o NAVCOMMSTA Resource Allocation Model - provided as a methodology example
- o The Enhanced Logistics-Over-The-Shore Model (E-LOTS) that is currently being used to support studies and other models in this area
- o A brief review of the area of mathematical programming systems.

4.1 NAVCOMMSTA RESOURCE ALLOCATION MODEL

The NAVCOMMSTA Resource Allocation Model was the result of a pilot study to develop and present a mathematical model relating the work performed by a NAVCOMMSTA to the input resources required, in terms of MILPERS, CIVPERS ceiling, and dollars. The model consists of a set of empirical equations based on data from existing reports, and a calculation procedure which has been translated into a computer program. With the communications and support tasks to be required of a COMMSTA as inputs to the program, the user can determine the personnel and budget resources necessary to support those tasks.

The model can be regarded as a form of nonlinear input-output (Leontief) model. The model first calculates the number of personnel required to support the NAVCOMMSTA communications tasks. The support tasks are then considered on an iterative basis analogous to the matrix iterative solution methods required for Leontief models prior to the availability of digital computers. The iteration resolves the additional support required for the support system itself.

The model is discussed here as an example of the kind of methodology that may be useful in computing support requirements for material/ transportation planning.

4.2 E-LOTS

The ORI Enhanced Logistics Over the Shore (E-LOTS) simulation model is an expected value model of a LOTS or LOTS-type operation, such as a Marine Amphibious Force (MAF) Assault Follow-on Echelon (AFOE). It begins as ships arrive off-shore and terminates when the last cargo module has arrived at its destination. The purpose of the model is to assist in the determination of the relative effectiveness of alternative resources in the ship-to-shore movement. In the process of accomplishing this, it also provides progress and completion time reports.

The model, which is in the "deterministic simulation" class, is a greatly improved version of a TRANS-HYDRO Study model originally developed by the Army Logistics Center, Ft. Lee, Virginia. The Army LOTS model was modified for greater realism, new features were added, capabilities were enlarged, and improvements for computer efficiency were made by ORI in support of the Joint Logistics Over the Shore (Joint LOTS) Test and Evaluation Program during the 1974-80 time-frame.

Subsequently, the model which has only a limited 5-ship capability, was greatly expanded to its present state now operable with about 35-40 ships (depending upon types). Cargo modules were expanded to reflect a 35 percent broader spectrum of pallets, vehicles, and general cargo. New lighterage and tractor-trailer units were incorporated to reflect large force capabilities and to represent current resources and operational characteristics.

The reporting format focuses on a series of cargo, ship hatchslot, lighterage, MHE, and tractor-trailer interactions. Ultimately, productivity in terms of short tons/hr, cubic feet (cube)/hr, and square feet (square)/hr are reported as model outputs.

Ships are introduced on a scenario controlled basis. On each ship preselected cargo items are located on various decks and in various hatches as appropriate. Hatches are opened at predetermined times and closed when totally emptied; the ship sails when all cargo has been off-loaded. Once all the ships have been emptied and the cargo is ashore, the run is completed and a wrap-up report is printed.

The model was originally designed to operate on an IBM 360/65, requiring 230K bytes of main memory, of which 78K was required for data. Subsequent expansion and inputs raised the requirement to 240 bytes and currently the simulation now requires about 250K with about 98K bytes required for data. The program is written in standard ANSI (x3.4-1966) FORTRAN. On an IBM 360/65, the execution time as on the order of 1.5-2 minutes or more, depending upon the size of the run. The model has also been run on a CDC 6600.

Currently, the E-LOTS model is being used on ORI's PRIME 400 computer. On a normal batch run, results have been available overnight; however, for runs involving 25-35 ships in the same batch mode conditions (low cost/low priority), a full day or so may be needed because of the size of the data base and amount of processing and printing.

4.3 MATHEMATICAL PROGRAMMING SYSTEMS

Mathematical programming systems are analogous in several ways to data base management systems:

- They are available for most processors as either manufacturer-supported or third-party software.

- They generally include a number of subsystems such as control languages, problem specification languages, algorithmic procedures and report writers.

- They vary widely in features, and are generally regarded by vendors as highly proprietary programs.

- They require significant effort to formulate and implement an application. In using mathematical programming systems this effort is further complicated by the need to consider the mathematical assumptions of the application and of the available algorithms.

5. SYSTEM ELEMENT AVAILABILITY

Development of the material/transportation planning system will require a combination of:

1. Off-the shelf hardware and software

2. Application-specific tailoring and interfacing of the off-the-shelf software

3. Application-specific system analysis and software development

4. Application-oriented research and development of mathematical algorithms and solution methodologies.

The costs and technical challenges of the application-related efforts (items 2, 3 and 4) are likely to far exceed the costs and technical issues related to the off-the-shelf system elements. The repertoire of off-the-shelf hardware and software over the next five years will likely be more than adequate to satisfy the possible uses of these elements.

6. RECOMMENDED INITIAL DEVELOPMENT PLAN

There are probably hundreds of people working for various agencies/sponsors to develop methods and systems for DOD-related material/transportation planning. Some of these efforts may be relevant to the system of interest here; at the same time, there may be gaps in needed activity.

In order to scope the activities required for development of the subject system, it is recommended that a survey be conducted examining the relevant capabilities/design of the present WWMCCS facilities and interviewing the following personnel:

1. The military command/staff personnel currently performing the activity that the future system is intended to support.

2. Sponsors of research into improved methodologies for supporting this activity.

3. Key members of the research/development community currently working on improved methods.

4. Vendors of potentially useful off-the-shelf hardware/software identified during the study.

Unless a clear requirement emerges for extensive cost minimization, it is not recommended that this survey address commercial applications.

The purpose of the survey should be to identify:

1. Existing system components and methods that are satisfactory in their present form. These components and methods should be considered as candidates for inclusion in the system.

2. Existing system components and methods which should be enhanced before they are included in a future system. These components and methods should be considered as candidates for enhancement.

3. Proposed methods that are recognized as useful but are not currently being used because of implementation problems (e.g. methods that require too much computing time). These methods can be considered as candidates for improvement using new computer technology or better mathematical techniques.

4. Recognized problem areas for which there are no existing solution methods. These problems can be considered as candidates for research into solution methods.

5. New ideas that arise during the dialogue between the survey team members and the interviewed personnel. These ideas can be considered as candidates for further investigation.

Following the survey, there should be a phase during which the candidate system components, methods, problem areas, and ideas are evaluated, prioritized, and assessed for cost and risk. The result of this phase would be a development plan for the desired system.

It is estimated that this activity could be accomplished by 2-10 people over a 6-12 month period.

REFERENCES

1. H. H. Barker, E. M. Sharon, and D. K. Sen, "From Freight Flow and Cost Patterns to Greater Profitability and Better Service for a Motor Carrier," Interfaces, Vol. 11, No. 6, December 1981, pp 4-20.

2. K. L. Stott and B. W. Douglas, "A Model-Based Decision Support System for Planning and Scheduling Ocean-Borne Transportation," Interfaces, Vol. 11, No. 4, August 1981, pp 1-10.

DATABASES and INFORMATION MANAGEMENT

Gio Wiederhold

Computer Science Department
Stanford University
Stanford CA 94305

9 September 1983

1. OVERVIEW

1.1 Description.

1.1.1 Basic definitions and terminology.

A database is a recorded collection of facts organized so that they can be computer processed.

A database management system is an integrated collection of programs to make such processing convenient and effective. Database management systems exist in generalized and application specific forms.

A schema contains meta-data describing the database itself. A schema is associated with a specific database and used by the database management system to control and maintain the database.

The combination of data, meta-data, and a database management system will be referred to as a database system. There exist database systems without a formal database management system.

1.1.2 Objective.

The objective of a database is to share data for decision making. In this sense the database provides a means of communicating information among distant functions and also among functions which are active at different times. The functions themselves will differ in objective and scope and will often place conflicting demands on the database.

A very difficult balance to achieve is the tradeoff between performance and flexibility. A large quantity of data increases the significance of performance factors. The variety of uses stresses flexibility. Multi-user operation impacts database integrity. All these desiderata have to be considered when selecting a database management approach.

Much data is collected and used for operational administration in a class of functions supported by routine data processing.
An expectation of database users is that the information held in a database can also be made available for decision making and planning purposes.

1.1.3 Components.

An operational database system includes data and program components.
The programs tend to be more general, and shared among multiple databases.
Components of the typical database system include:

1. Data representing facts of interest to the enterprise being described.

2. A schema describing the data being stored.

3. Archival and logging data in order to provide audit and recovery capability. This will protect all information stored currently or in the past within in the database.

4. Programs which manipulate the database using the schema information. These programs provide the interface for the programs which update or retrieve data from the database.

5. Programs which interpret user commands and translate them to arguments for the programs which manipulate the database. These programs provide the interface for the users who update or retrieve data from the database in an on-line environment.

6. Programs which translate the database description into the internal schema. These programs are tools for the database administration.

7. Utility programs to maintain the databases.

8. If the database is distributed over multiple machines there will also

be programs to schedule query execution over the mutliple machines and control the communications which are required.

9. In systems using statistical or knowledge-based approaches there will be modules to draw inferences from the meta-data and the data stored in the database. The effect is to reduce voluminous data to a smaller amount of manageable information.

Different database management systems have given different levels of stress to the nine elements described above. At times the reason for the difference in stress is historical; many current database management systems are generalizations of advanced data-processing packages of the past. In many instances database management systems reveal the experience of their writers. No database management system today satisfies all desiderata, from flexibility to performance.

Dealing with database management systems that are not fully adequate for some task is frustrating for the users. The database administrator is typically limited to making changes that the schema provides for. The actual software currently available for database management systems is not organized so that good modules from one database management system can easily be made to work with modules from another database management system.

1.1.4 Transactions and Queries.

In discussing databases we must consider how they are used. Three types of usage can be distinguished:

- o databases are manipulated by programs which process input, output, and database files. These programs may run in batch or timesharing modes. This use is where databases mainly generate reports.

- o databases are manipulated by users invoking pre-written transactions from on-line terminals. This use is common where the database is used to share informtion, as in airline and banking applications.

- o databases are manipulated by users interacting directly with the database, typically via a query language. This type of usage is associated with decision making.

We will elaborate the latter two types of use further, because some of the uniqueness of database requirements derives from them.

Transactions.

The dominant usage today of large operational databases is through transaction programs. Transaction programs are designed so that they carry out single but complete operations on a database. They are invoked on-line by a scheduling program in response to simple user command codes. An on-line transaction typically requests formatted input from a screen and generates screen output as well as printed reports when needed.

Transaction programs are written either in a special or an extended general purpose programming language and execute relatively high-level operations in order to retrieve or store data into the database. In these languages data elements are referred to by name rather than by address. Data elements are often of variable length and the transaction program is not aware of the detailed physical structure of the data being worked on. Transaction programs typically are limited to several thousand lines in length and execution times are a fraction of a second. These constraints simplify transaction scheduling.

Systems which support transaction management often provide integrity maintance services which assure that, if a transaction cannot be completed, all effects of partially completed transactions are erased from the database. This assures, if the transactions themselves behave correctly, that a database will not be damaged due to failure of individual transactions.

Query access.

Another mode of operation of a database is direct inquiry via a query language. Query languages come in many modes. In the simplest mode the processor interrogates the user in order to construct a query. A menu is presented on the screen and the user is lead through a narrow set of choices. On the other extreme are natural language interfaces where the user is completely unconstrained in the querying. The processor has to produce cooperative responses to cope with wrong assumptions about vocabulary, structure, and content of the database system. In update activities the climate tends to be considerably more restricted and many databases are only updated by transaction processes or via batch programs.

1.1.5 Architecture.

The terminology in the area of databases is becoming much more consistent and well accepted than it was a short time ago. Nevertheless a fair amount of confusion remains when system architecture issues are explained to outsiders and some fashionable terms are used with little discrimination. One set of terms, namely relational', hierarchical', functional', and network' is used to refer both to the architecture of database interfaces and their implementation.

Interface architecture.

A relational interface indicates that the database can be viewed as consisting of rectangular tables. The user specifies at query time operations which relate records in these tables to each other based only on attribute names and the values found in these tables. The relational interface is formal and relatively easy to learn. The basic interface does not provide for update constraints and does not provide transitive closure, i.e., looping until a condition is met. Two forms of the relational interface exist. In the relational calculus the query is stated in the form of expression which ranges over the database. In the relational algebra the result of the query is specified as a procedure using database operations on the tables of the database.

The relational calculus is frequently translated into relational algebra as a step towards query execution. If quantitative parameters of the database and its attributes are known a considerable amount of optimization can be performed during this query translation. Programs which specify queries in the relational algebra can also be optimized as a whole. Individual steps in the relational algebra provide less scope for optimization.

The hierarchical interface assumes an underlying model for data that have hierarchical dependencies among each other. Queries following this hierarchical structure can be stated simply. Queries which do not follow the hierarchical structure of the model which was assumed for the data, can be awkward and often impossible to answer.

A network interface to database assumes a structure that may be arbitrarily connected. The connection is specified in the schema. At the simplest level the data required to produce the result for a query is collected by navigation through the database, one point referring to other related data points. The navigational process may either be specifically described by the programmer or may be automatically generated by translation from a higher level language. When there are multiple candidate connections for a query between some network nodes the path between these nodes has to be decided. Often the choice is easy and can be automated, sometimes it has to be specified.

Functional languages use a specification among the network nodes which relates these objects in a functional form.
The specification, kept in the schema, can resolve much of the ambiguity inherent in networks and make translation from higher level languages to the navigation level much more feasible.

Implementation architecture.

Databases may be centralized or implemented on multiple, distributed computers. In either case the database at one location can be characterized by an implementation architecture. The implementation of databases is often defined in terms which correspond to interface architecture terms: relational', hierarchical', and network'.

Systems using these implementation technologies are not necessarily restricted to the interfaces which have corresponding names. Given our understanding of mapping between these implementation structures it is possible to provide the relational interface for any of the implementation structures, given that certain constraints are made and that information loss during the transformations, which lead to the alternative implementation structure, is avoided.

A relational implementation typically implies a simple mapping of data into simple fixed record-length files. When rapid access to some records is required indexes may be used. If access by any attribute must be fast then indexes for all attributes must be maintained.

A hierarchical system typically organizes its data physically as a depth-first-tree in storage. Sometimes only the access structure is hierarchical, and the data are kept in simple files. Very rapid access is provided when data is processed in the pre-order' sequence which matches this layout.

Network databases include cross references from one record entry of one record type to record entries of other types. These references may be symbolic, indirect, or direct. Maintenance of cross-references requires care and incurs costs at update time. Use of references can be very beneficial when the database is being queried.

Symbolic references are the most flexible and depend on structures similar to indexes. Indirect references depend on intermediate tables or hash functions. Direct references provide single step access but confine the resulting database into an extremely rigid structure. Sophisticated data structuring and reorganization functions can mitigate the effect of rigid structuring.

Distributed databases.

Databases may be distributed over multiple computers located at distinct sites. At the communication network level all participating databases must present a similar interface. Primitives on the level of relational queries appear well suited for such a communication protocol. Due to the relatively high cost of communication, data may be replicated at multiple sites in order to enhance query performance.

Replicated data creates concern for consistency and hence synchronization. A related issue is the degree of autonomy accorded to the distinct nodes. The motivation for distribution often includes a desire for autonomy; however, rules for maintaining consistency among replicated data in a distributed database may conflict with such autonomy.

Most distributed database systems today have been hand-crafted and operate on identical or similar computers. Developmental work has included providing front-ends processes to existing databases in order to match a shared communication protocol.

1.1.6 Summary.

In summary, when we are dealing with databases we are dealing with products which have to satisfy multiple objectives and hence often include compromises, especially in terms of the level of performance allocated to the distinct objectives.

Database systems support conceptually seperatable functions.
In many systems we can identify perhaps ten such functions.
Most functions have a number of alternative design choices.
Some of the design choices are associated with well-understood concepts and influence greatly the interface and performance of the database system.

Most of today's database systems are highly integrated and the modules which support these functions are not very distinct nor are they interchangeable. Work in distributed systems is providing an impetus for greater modularity of database management systems.

1.2 Discussion

1.2.1 Examples

The attached appendix (B) lists a large number of operational and developmental database management systems, and identifies their principal features. There are still many operational databases which do not use a database management system, but have been built using file access programs and semi-systems programs to implement the functions required for database operations.

The major commercial databases which appear in this appendix have reached a high level of functionality and reliability. Associated with that success is an increase in complexity and limitations in portability. Maintenance cost, both of the database management systems themselves and of major database applications that use these systems, tend to be high.

Maintenance is required when there are changes of requirements or when there are changes of equipment on which the database management system is operating. Equipment changes may be necessitated by a growth in quantity or scope of the application.

High levels of performance can be reached in many commercial database systems when the administrators and users are knowledgeable. Design tools to configure optimal databases are becoming available but require information on the expected usage distribution of updates and queries to the database. Optimization is often limited to either aggregate optimization or satisfying requirements for individual performance. These two types of optimization may conflict with each other.

1.2.2 Performance metrics.

Performance in a database is a question of the balance of the requested activities, and the capability of the database system architecture, the database management system, the system software, and the system hardware, to respond to these demands. Since compromises are typical in databases, we do not find that there is simply one approach which is best in general.

Database activity.

The variables which determine the activity of the database are

- The size of the database categorized by record type.
- The number of retrieval requests, categorized by type, directed to the database.

- The number of update requests, categorized by type, directed to the database.

The usage of the various types of retrievals and updates typically has a Zipfian distribution. This means that a relatively small number of types, say 10% to 20%, account for a large fraction, say 80% to 90%, of database usage. This means that performance prediction for realistic databases is feasible by concentrating on these high volume transaction types, although a fair amount of analysis work may be required.

Database operations.

The controlling variables for the performance of the database are

- The organization of the files in the database.

- The means used for making cross references among files in the database.

- The performance of the system software.

- The performance of the hardware.

The principal hardware parameters in turn are

- CPU speed, especially the execution rate in moving data.

- Disk seek-speed.
- Disk rotational latency.
- Disk transfer rate.

- IO bandwidth limitation.
- Communication bandwidth limitation.

In many current database management system implementations the cost of executing the required system commands for buffering, resource allocation, synchronization control is so high that CPU speed is an important parameter in systems performance. We consider however that the disk parameters and, in distributed systems, the communication parameters lead to more fundamental limitations. Current advances in technology have done more to increase storage capacity than to increase the performance parameters of disk storage devices.

When the operational system parameters listed above are known, performance metrics for the primitive operations from which queries and transactions are composed can be estimated.

- o Retrieve a single element using random access to disk storage.
- o Retrieve a successor element according to a known organization.
- o Update an element at a known location.
- o Delete an element from a known location.
- o Insert an element at an appropriate location.
- o Restructure or reorganize database.

The importance of low-cost successor operations must be stressed. The most interesting information-generating application are not based on a single element which is retrieved but require a large number of related elements. Acceptable costs for single element random retrieval rapidly become unacceptable when multiplied by element counts of a thousand or more.

Retrieval versus update.

High performance data retrieval is obtained by constructing access paths into and within the database structure. Access paths are built by increasing redundancy and by replication of data. These paths require maintenance to assure consistency and integrity of results.

The basic data update operation is typically less than two times as expensive as the primitive retrieval operation. Associated with updates, however, are typically a number of retrieval and secondary update operations in order to maintain all affected access paths. The effect is that updates may be a factor 10 more costly than data fetching operations.

In order to maintain high performance in a database environment which is frequently updated, periodic reorganizations may take place. Reorganizations move some of the cost of maintaining an adequate performance level from the time of the individual update operation to a time which is more suitable. Reorganization frequencies range from daily to yearly. They may be associated with other periodic processing functions in the database.

Use of performance metrics.

When the cost of the primitive operations within a database is known, the performance of transactions on a database can be predicted with engineering accuracy, say within 20%. If many simplifications are made the predictions may vary by a binary order of magnitude. Such predictions are still greatly preferable to situations where no attempt to predict performance is made and at times products are delivered which fail their objective disastrously.

1.2.3 Level of Effort

The level of effort expended to bring current major databases into operation has often involved tens of man-years. A continuing effort goes into system maintenance as well, especially as the operational environment changes and users develop new requirements. Not all of this effort is necessarily visible in the final product. Lack of planning, analysis, and experience has made that in many datbase systems major sections have been rewritten several times. The cost of rewrites is often high, mainly because no formal interface specifications have been developed and evaluated prior to implementation.

The use of a database management system can bring a database system much more rapidly into operation. Simple applications may be operational in a week, although major applications will still require major efforts in order to deal with their data input and result reporting needs.

The effort expended into developing a database management system itself is least a binary order of magnitude greater than the effort to write a database system for a specific set of applications. Much effort is expended by the initial users as well. On the other hand, broad acceptance of a database management system has a high positive leverage.

Also in the development of a database management system careful planning and experience can make a major difference. Those database management systems which have grown incrementally with the applications appear to need continually much support. Those database management systems that were written more in the abstract appear to need less ongoing development support, but are also more constrained in growth.

Portability of database systems between computer types has only been feasible where portability was an initial design goal. Where database systems have been rewritten without conceptual changes the efforts have been tolerable. One recent such rewrite was the adaptation of the research and development system at IBM - System R - written in PL/1, to the commercial version SQL/DS written in the IBM system programming language PL/S. I would guess that this effort took about two times ten man-years?

It seems reasonable that most modules of a well-specified database management systems can be written within a span of a year by small teams of two to three people. A database management system may comprise on the order of ten such modules. As far as I know, no system made completely out of modular sections exists today in the commercial environment. Our estimates are based on experience with modular aspects of research and development systems and we consider in this estimate those modules that provide support functions rather than those which comprise a research contribution.

The effort to implement a database management system depends greatly on the availability of an adequate file system and convenient access to that file system using the programming language to be employed. At the same time an adequate file system can be better designed if database requirements are taken into account.
Many current file systems do not take database management system requirements into account, are optimized for easy use by programmers, and provide an unsymmetric and sometimes idiosyncratic interface to the database management system.

A substantial amount of coding in database management systems is devoted towards file management, sometimes bypassing facilities accessible to programmers, At times database management systems replace existing file access services with other facilities and sometimes transformation programs are placed between the database management system and the file management system.

A disadvantage of this dichotomy is that it can be difficult for database management systems to utilize data stored in file structures defined independently by programmers and it can be equally difficult for programmers to safely interact with file structures used by the database management system. We believe that careful design can avoid this dichotomy. This opportunity exists today in ADA because, as far as I know, no packages for advanced file management have been specified.

2. FUNCTIONAL REQUIREMENTS.

2.1 Introduction

The functional requirements for database management systems can actually be stated relatively simply. They are best divided into categories:

- o Logical requirements
  Here we are concerned with functionality and user interfaces.

- o Physical or configuration requirements
  Here we are concerned with adapatability to operating systems.

- o Performance requirements
  Here we are concerned with internal functional capabilities and the flexibility to configure the system to support the usage patterns.

Related to these requirements are requirements for adequate

- o Reliability and backup.
  Here we consider the utility services required for database maintenance.

In the modular approach we envisage several levels of requirements could be supported for each category. For instance, backup requirements differ greatly for, say, financial transactions versus signal-data processing. In the latter case, historical data has often little value and is very voluminous at the same time. We will discuss issues of level in more detail with the various requirements.

It will be desirable to support the same logical, or user, requirements in each of the categories; although performance may differ so greatly that in practice some of the logical functions could not be carried out in a timely fashion if an inappropriate level of implementation module is used.

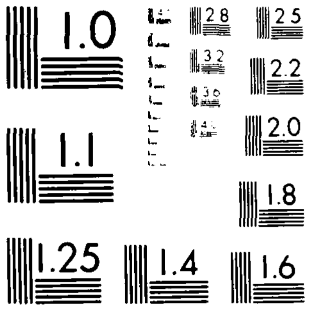
2.2 Logical Functional Requirements

For programming level access in this category we require the ability to read, write, and modify single or sets of data elements. The data elements should map directly into the data types provided by the

AD-A142 570 WIS IMPLEMENTATION STUDY REPORT VOLUME 3 BACKGROUND INFORMATION(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T H PROBERT 01 OCT 83 IDA-D-51-VOL-3

2/6

UNCLASSIFIED IDA/HQ-84-28344 MDA903-79-C-0018 F/G 17/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

source language, here ADA.

A key' specifies an attribute and a value for access. Serial access uses keys and must be independent of physical sequential order.

We will consider for programming-mode access in this category two levels:

- o Data-processing level
- o Database management level

Furthermore, we have to consider at the database management system support for user query languages.

Associated with all of these operations it a database reference' type which can be used to recall a position within the database. This reference data type will be heavily restricted, no conversion or computational operations would be applicable to it.

2.2.1 Data-processing level

At the data-processing level the users themselves are aware of the logical mapping of data-elements to records and the placement of records into files. No schema is interposed between the user and the access function, although the physical mapping may be complex.

The statements used by the programmers have three components:

1. a file identifier, established by an OPEN statement.
2. a reference variable, which identifies a current logical position in the file.
3. a data variable, typically a record stucture.

A language which provides examples of such statements for file usage is PL/1. Unfortunately in IBM's PL/1 implementations the precise semantics of the execution of such statements is affected by the physical storage types chosen. Such a dependency is not necessary.

We will illustrate the applicable PL/1 statements below, since random file access is an essential module within a database system, although a database management system will not provide these statements at its functional user interface. We do not find these statements ideal, especially since their action is also influenced by options given in the OPEN statements for the files. The option combination DIRECT and UPDATE enables most of the actions. The need for for KEYFROM is not clear versus the simpler form KEY. The DELETE statement behaves inconsistently, it was modified in our implementation {PL/ACME} to prevent disasters in on-line operation.

```
   Read file randomly by a single reference attribute:
     READ FILE (filename) KEY (key-expression) INTO (variable)

   Read file serially according to the reference attribute:
     READ FILE (filename)  KEYTO (key_variable)  INTO (variable)

   Append to the file according to the reference attribute:
     WRITE FILE (filename) FROM (variable)

   Insert into the file given a reference attribute:
     WRITE FILE (filename) KEYFROM (key-expression) FROM (variable)

   Update the file given a reference attribute:
     REWRITE FILE (filename) KEY (key-expression) FROM (variable)

   Update the file according at the last reference read:
     REWRITE FILE (filename) FROM (variable)

   Delete a data element from the file given a reference attribute:
     DELETE FILE(filename) KEY (key-expression)

   Delete the last prior data element read from the file     * added
    { DELETE FILE(filename) RECORD }

   Delete the entire file
     DELETE {ENTIRE} FILE (filename)       * ENTIRE added for safety
```

PL/1 Language Record Access Statements

The concept of a next' record has had its basis in the conventional sequential file organization. In these files however next is a logical, key-based concept, and next records may appear at arbitrary physical locations in the file space, especially if the file has been created or updated with many random insertions and updates. Updates cannot necessarily be rewritten in place when the variables include variable length structures.

A simple implementation level for such files, restricted to fixed record sizes, exists on most computer systems, in order to support requirements posed by COBOL and extended FORTRAN compilers.

These data-processing statements can provide a foundation for database management systems. Effective database management systems and database systems have been written in PL/1 and, more importantly, ported to DEC VAX and ALTOS microcomputer equipment when PL/1 compilers for these machines became available.

2.2.2 Database management level

We emphasize again that the statements shown above are not database management system statements. For databases we expect to access individual data elements, selected through a schema, according to any of multiple atributes (rather than a single key), and access data from multiple files,

At this level a set of statements is also needed to maintain the descriptors in the schema, the meta-data of the database. For most users the descriptive information is obtained simply by including a WITH-DATABASE(...)' statement in the programs.

However, support for a database program management function has to be provided which assures that

a. Any routines which obtain a schema which has been modified are automatically recompiled.
b. That programs that use a schema are logged into a control system in order to provide the possibility of verification of schema changes.

Automatic management of compiled databases code is a feature of IBM's system R and SQL/DS system. The alternative is to interpret all statements which refer to the database. Since database access is often slow compared to processing times, the cost of interpretation may be bearable in many situations.

Programs which include database access statements to be compiled may be processed through a pre-processor which transforms the statements into calls to the database management system, using the schema information to select the most appropriate calls. The use of a preprocessor can avoid extending the requirements for an ADA compiler.

The issue of compiling versus interpretation of a schema is not an issue of level, but rather an implementation choice. The specifications for the database management language must be such that neither compiling nor interpreting schemes are precluded.

The schema language.

The schema language specifies the logical appearance of the database. A user application needs only to include a reference to the existing schema. A user's schema is also best restricted to purely logical aspects, and may be further constrained to a subset, or subschema of the database.

A simple schema language includes

1. Specification of all logical record types in the database

2. Specification of all attributes in the database. Included are at least
   - o name, the user's reference
   - o type, i.e., a refence to a domain

3. The mapping of attributes to record types

4. The constraining connections among attributes in the records

Ideally, at this level only logical constraints are specified. For instance, selection of some attribute to control physical ordering is an option of the physical specification.

An example of the major parts of the CODASYL 1978 logical and physical schema languages is given in Appendix A. We should note that, as far as I know, only one database management system has adopted the 1978 specification, and all major current commercial systems use subsets of the 1972 specification, which mixes physical and logical requirements.

Relational database management systems have very sparse schema languages, since these systems have minimal semantics, and rely on the formulation of the queries to produce correct results. The most elegant schema language is perhaps that available with the CII Socrate system, using a PASCAL-like syntax.

The schema language itself is best processed through a schema language compiler. This compiler changes the representation of the schema from a format that is optimal for human specification into a representation for rapid decoding at pre-compile or interpretation time.

Database manipulation language.

Database manipulation statements of distinct database management systems vary greatly in form or syntax, but some basic features can be recognized in most of them:

1. The definition of a set of tuples or logical records. The set may be specified by name, may use set operations over the components of the database, or be identified by a dependency from some other database component. Set membership may be constrained by selection of tuples satisfying certain conditions.

2. A cursor. To give access to members of this set within a programming language which cannot handle sets of arbitrary size, a cursor is defined. It is manipulated to provide access to one selected record at a time.

3. Projection of the set or field access. Specific attributes of the set to be retrieved are identified; or data elements from the current record can be manipulated individually.

Although we have described the functions in terms of retrieval, update of a set may be supported symmetrically. Here however many restrictions prevail. For instance the set selection is typically limited to an entitre untransformed set or one tuple at a time of such a set.

Query languages.

For interactive applications the logical requirements are quite different. Here we envisage a front-end query compiler which translates queries using the combination of a generalized query-processing dictionary and a dictionary which is specific and integral to the database. The result of the query processor is intermediate language program which is interpreted by a set of simple programs which operate on the database.

2.3 Physical Functional Requirements

The physical configuration of the database determines to a great extent the attainable performance of the database. It is, however, constrained by the logical specification: all logically valid data combinations must have a corresponding physical representation. This function specifies hence the mapping for all record types, attributes, and connections among those attributes to physical storage. Also the methods to deal with file growth and physical re-allocations need to be specified.

Desirable file organizations, creation of indexes to records, or links between records, hashing transformations, and storage allocation for growth of the database can be specified here. A minimum requirement for current databases is the availability of an indexed-sequential file access method. However, symmetric performance can only be attained if file methods which provide access by more than one key are available.

The functional requirements are specified in a portion of the schema language which may have restricted access. Two approaches are feasible:

1. Specification of goals;
   We may say, for instance, that random access to record R using attribute A is frequent.

2. Specification of the implementation;
   We specify that record R is to be indexed using attribute A.

The former approach, a goal-oriented specification is uncommon today. Portability could obviously be enhanced, since database management systems could take advantage of those physical facilities available on the supporting software and hardware systems. In most cases the database administrator has to translate the implied objectives from one physical specification into a new physical specification.

The physical specification is concerned with the allocation of logical programming variables and records to actual physical storage. This specification also determines the the access path best suited used to retrieve or update the data.

Options to increase performance include

- o index creation for certain attributes and attribute combinations.
- o providing ordered storage of records by some attribute.
- o providing hashed access to records by some attribute.
- o clustering of records having common attribute values.
- o implementation of logical connections by pointer references between records.
- o clustering of connected records.

All these choices can be made without affecting the logical correct operation of a database. The performance differences cannot be hidden from the user.

Views.

In a multi-user environment it may be necessary to map multiple overlapping logical descriptions into one integrated description. The individual user may be limited to his or her original subset, or some other limited view' of the database.

Fragments.
In advanced versions it should also be possible to do record partitioning and recombination.

Record partitioning consists of mapping of logical records into fragments: multiple smaller physical records. Fragments may reside on distinct devices, including devices of different performance or also on different processor nodes.

Integration of records can combine distinct records or their fragments, obtained from partitioning, into larger physical units for economy of
dmacs
access.

Record mapping options of the schema.
We hence see that the schema has to specify several levels of logical to physical mapping.

- o One-to one logical to physical mapping.
  All users which acquire record will have access to identical copies of the entire record.

- o Users specific record subsets.

Here we view the database records as being integrated from the requirements of many users; each user's view however is restricted to those data elements of the record which are appropriate to the user and as such specified in the user's subschema.

- Complex logical to physical mapping.
  Here the database records are created out of record fragments which may be distributed over multiple physical records. Fragments may also be obtained by unpacking them from combined physical records.

The mapping of logical to physical records is THE primary tool to control access and performance to databases. Many database mamagement systems today support only a one-to-one mapping or a limited one-to-n hierarchical mapping. Those systems, in effect, put the mapping responsibility on the user.

2.4 Performance Requirements

Performance requirements are specifications at a higher level of abstraction. They are related to goal-oriented physical specification, but include quantitatively specific values.

Performance requirements first can be stated as the required response times for query types and transactions.
Other performance specifications may include bounds on storage capacity, memory capacity, or functionality of underlying software.

At the current state-of-the-art performance requirements are applied during the design process in order to make reasonable design decisions, typically using the physical specifications described above. Subsequently they are used for acceptance testing of the software.

It is conceivable, but not yet possible, at the current state-of-the-art to generate the physical specifications automatically from performance requirements. The architecture of a DBMS should not prohibit such advances by providing control of physical specifications to users who are typically only concerned with the performance of their subset of the database functions.

2.5 Maintenance and reliability services.

An important motivation for the use of database management systems is the comprehensiveness of the support services included in such systems. The increment of effort from writing a functionally adequate database system to one that protects its contents in all kinds of adverse circumstances is major. We cannot ignore this aspect.

The level of backup should be selectable for each of the physically physically stored data elements in the database.
Options to be considered here

- o For transient data, simple copy logging of inputs.
- o For audit trails, complete before- and after-image logging.
- o Transaction-thread logging for databases with full recovery.
- o Transmission of backup data to remote computers in order to avoid node vulnerability.

The configuration choices to be made here are typically set as installation parameters. Such parameters could be usefully included in the schema specification.

Associated with these services are utility packages to effect database recovery.

2.6 The technical challenge.

2.6.1 The general state.

We find ourselves here at a boundary of technology. There appears to be an adequate understanding of the problem, and adequate solutions to all of them. At the same time, we have no acceptable example of a satisfactory solution to the entire database problem, and we can expect that better solutions for each of the subproblems will be forthcoming.

In the terms of this report, it appears unclear whether databases can be fully integrated into ADA at the 1986 or the 1989 phase. The demand for data-processing services appears significant, and a lack of such services will lead, perhaps even before 1986, to the development of packages which will satisfy specific needs, but those packages will use a wide variety of file conventions and formats.

It appears feasible that a complete set of external package specifications can be developed now, and perhaps primitive implementations be completed for many of the modules. Such an approach may channel the efforts whoch will undoubtedly be expended in this area into a stream which will consider consistency as one of its objectives.

In order to achieve such a growth path we put forward the following general requirements for ADA package interfaces in the database area.

2.6.2 The schema.

We will first consider the schema. A question here is if the internal format must be specified to achieve commonality and portability. Our assumption here that this is not necessary, and that an object-oriented approach will serve the requirements well.

- o Specify a standard and extensible schema language using ADA-style data typing parameters.
  Within the schema separate the logical and physical specifications.

- o Provide the capability to specify performance requirements even though they may not be automatically processed.

- o Provide a language interface specification to create, update, and store database specifications given in the schema language.

- o Specify language facilities so that users can retrieve information about data stored with a schema.

An expandable schema specification is needed to provide opportunity for growth. In more advanced schemas we expect to also find information about the semantics, ownership, and even history of the data.

2.6.3 File access.

Next we need specifications for primitive operations on files. We believe we have to be able to support data-processing as well as database management systems. All records acceptable to ADA should be acceptable to the data manipulation language and vice versa, given that the schema definitions are appropriate.

These data-processing statements can use basic facilities from a schema for record definition. An example, but not a model, for such basic facilities exists in the file definition section of COBOL.

- o Provide record oriented data manipulation statements including
  FETCH,
  GET_NEXT,
  INSERT,
  APPEND,
  DELETE,
  UPDATE.

Data types which can be manipulated with these statements should include not only current ADA primitives, but also

- o Variable length strings, according to a standard ADA package definition.
- o Reference, to database elements according to a database package definition. This datatype can only be moved, copied, and compared by ADA statements.

The database manipulation processors should use this interface for all their file access. These statements must be supported by file accessing programs. There are many choices here, and it is not neccessary to specify how the access is to be implemented, but only its functional behavior. We will list some of the choices as an existence proof.

File access alternatives.

Alternatives to be considered for file access support include

- o Multikeyed access using B-tree technology.

- o Expandable (linear) hashed access.

- o Access methods exploiting optical disk technology.

These variants should remain invisible to the user except in terms of procedure performance.

2.6.4 The database manipulation language.

The database manipulation statements to be included require more thought than given here in this initial report. As a basis we can propose a relational algebra. A relational algebra, in this context, has the following advantages and disadvantages:

1. A: Implementation is simple and unambiguous.
2. A: Some impressive algebra based systems are in operation on large (Honeywell) and small computers (IBM PC).
3. D: A programmer can create extremely ineffecient programs, by inadvertently using large intermediate sets.
4. A: A programmer, especially given access to tuple-identifiers, can create quite optimal programs. For well-understood applications such programs will outperform automatically optimized programs.
5. D: Less research has been performed on automatic optimization of relational algebra programs than on calculus based systems, although there appear to be no fundamental reasons for less optimal results.

As an initial proposal we suggest the following functions, to be combined into programs using extensions of ADA program syntax:

o PROJECT(recordtype BY(attribute-list))

o SELECT(recordtype BY(attribute-value-list))

o JOIN(recordtype1 BY(attribute-list), recordtype2 BY(attribute-list))

o UNION(recordtype1, recordtype2)

o INTERSECT(recordtype1, recordtype2)

o DIFFERENCE(recordtype1, recordtype2)

o CROSSPRODUCT(recordtype1, recordtype2)

reference generating and manipulation functions:

o REFERENCES-OF(recordtype)

o UNION(references1, references2)

o INTERSECT(references1, references2)

o DIFFERENCE(references1, references2)

and a set of tuple-attribute retrieving functions:

o TUPLES-OF(references)

o FROM-FIRST-TUPLE(references, attribute) o FROM-LAST-TUPLE(references, attribute)

o FROM-NEXT-TUPLE(references, attribute)

o FROM-PRIOR-TUPLE(references, attribute)

Not included in the list are statements required to support multi-user operation. Such statements for instance define transactions and set locks. The ability to identify tuples uniquely by reference can provide a basis for interference checking.

We envisage these statements to be used by programmers working on major database systems and within transaction programs. Most users will not see these statements.

Implementation alternatives.

The database manipulation statements sketched above may be either interpreted or compiled. In a system oriented towards production only compiled use may be available, while in a system oriented towards nigh degree of interaction only interpretation may be available.
In general however the choice should be made by the user explicitly or implicitly.

2.6.7 Query languages.

Direct, on-line users will either interact with these transactions, or use query languages which generate these statements in response to the queries. Implementation of a functionally complete relational calculus system is nearly trivial given these statements. The automation of the optimization required for high performance relational calculus systems remains a challenge. The ability to manipulate tuple identifiers or record references makes such an optimization feasible.

2.6.8 Performance.

The performance of the database system is bound by the capabilities of he file systems which support it. In our experience database systems perform between 1.2 to 5 times as slow as programs which are written for the same applications and which use file systems directly.
The overhead of the database management system is typically warranted in terms of reduced cost to bring applications into operational state,

reduced cost to maintain the database, and improved reliability.
In some instances, the dominant reason is the possibility of sharing data among users.

There is no reason why an ADA based database ·nagement system should not perform on the desirable side of this range.

2..6.9 Summary

The technical challenge in bringing a database management system under ADA into operation are concentrated in the area of design.
We are confident that an acceptable design, augmented with basic functional modules, will lead to more sophisticated implementation efforts at many sites.

In order to achieve acceptability and reliability we assume that a design has to be highly modular and that over time muliple modules will be developed to carry out the same function with different bounds of performance parameters. Such systems do not exist today although we can identify systems which have examples of the kind of modules which we envisage.

3. MODULAR CASE STUDIES.

Since we cannot identify any system which is adequately satisfactory we will consider specific modules as a basis for this report.
We will first consider the candidate modularization. For each module we expect to find several implementations, although an initial development will probably be limited to one module of complete functionality, but low performance.

We will cite with these modules some current examples or relevant development work. This list is certainly not exhaustive.

3.1 Modules.

A suggested modularization includes four modules (identified as M1 through M4) to manage database resources, two modules which provide access to the schema for programs and the database administrator (M5 and M6), three modules (M7, M8, and M9) to carry out operations requested by the user, and four internal modules (M10 through M13) which carry out support functions. In certain environments some of these modules will be null.
A sketch relating such modules, from an earlier presentation, is attached as Appendix D.

3.1.1 Resource Management Modules.

We see four types of resources which have to be explicitly managed within a database management system: data, meta-data, transient memory, and archival storage,

M1. Data files.
A file access system maps access requests, which are stated in terms of file names and record references or attribute values, to operating system requests.

We expect that the operating system will provide access to computer storage capability based on a name and a relative address.
This system will allocate the physical records to blocks in storage, provide referencing and dereferencing capability to these records, and provide the capability to trigger other subsystems on request.
For instance the backup maintenance subsystem may require trigger whenever a record is changed.

A suggested implementation example is provided by FLASH, described in Appendix C. A number of B-tree systems are available in the personal computer market, these are typically restricted to fixed-length records and fields.

M2. A schema-table manipulator.
The schema contains the meta data which is the key to the operation of the database management system involving multiple files.
A schema contains many types of information

- o information which is global to the entire database management system
- o information which is related to the conceptual relations into which the database is decomposed,
- o information about the files into which the relations are physically mapped,
- o information the logical records which are presented to the users programming interface,
- o information about the physical records which comprise the database files,
- o information about the attributes of the individual data types which make up the user's records,
- o information about the fields and the representation used to store the data within the database.

Facilities to be provided by the schema-table manipulator include the retrieval of data descriptions, the update of schema entries, and triggering of file reorganizations and respond to changes in the schema entries.

The schema manipulator functions are used by the schema language compiler, and passively by all other operational components of a database management system.

Information kept in the schema may itself be stored using the file access system. Such schema systems are being develcped by Roussopoulos under a NASA contract, but such techniques are now used within IBM's system R on large machines and by Pacific Software's Sequitur as an example of a micro-based system.

M3. Buffer management subsystem.
An important resource of the database management system are the buffers which are used to collect and assemble blocks containing data records in core storage. Buffers are used to collect data for transaction processing and collect transaction results which can be committed into

the database when the entire transaction has been successfully completed.

Effective use of buffers provides rapid access to data which has been allocated to be stored together and thus permits exploitation of the locality directives stored in the physical description portion of the schema. Typically each open file requires a small number of buffers, say from two to five. Additional buffers can enhance considerably the performance of a database management system.

Buffer managers can be written at many levels of sophistication. In a multiuser operation buffer management carries on additional functions to assure synchronization of requests from muliple users. If the scope of locks is single records it is neccessary for users to share buffers when they are sharing files, since otherwise access conflicts to disk could not be resolved.

Keeping of information in backup buffers can provide older, but consistent copies of data to users which would otherwise conflict during simultaneous access to the same data.

Buffer management schemes are included in all multi-file systems. IBM's CICS and IMS have fairly complex schemes.

M4. Archive resource management.
An important aspect of commercial database systems is the management of backup files. Backup files may include the following components:

- o Before images.
  Copies of the data which existed in the stored database before any modification.

In some concurrency schemes, for instance ADAplex, such before images are retained actively in order to permit concurrent transactions to proceed while other transactions are affecting the database.

- o After images.
  Redundant copies of data stored into the database are written on backup devices in order to provide direction capability in case the primary write operation fails.

- o Transaction thread.
  In order to provide an audit trail for manual or automatic correction of errors found in the database either due to system, operator, or data entry error, a log may be kept of every record accessed for read or for write during any one

transaction.

- o Query text.
  Recording of the input to a transaction or a query can provide the ability to restore a database which had to be recovered from an earlier copy.

- o Response text.
  Storage of the response can provide a verification that earlier output was correct and can provide the backup for a retransmission if the communication failed without having to re-enter and re-execute the transaction and possibly introducing new inconsistencies during replay.

The extent to which backup is required varies greatly from application to application. In many scientific settings today, such backup provisions are effectively null. However, in commercial environments they are typically quite complete and experience has shown that serious hardware, operational, and programmers' errors can be recovered using backup facilities.

The backup module may use a file access system for its storage function. Frequently magnetic tapes are used for archival storage although in modern designs, magnetic disks and, we expect soon, optical disks can be used to serve the archival function.

Examples of backup and recovery modules exist with all major database systems. Major examples are found within IBM's IMS, Software ag's ADABAS, and Honeywell IDS II.

3.1.2 Schema access modules.

We now consider the operational modules which are associated with the schema. In general these modules are hidden from the user.

M5. A schema interpreter.
In order to provide convenient and consistent access for programs and database administrators which use schemas, a module to interpret the schema has to be provided. Since we expect a schema to be extendible, such a module must produce reasonable default outputs when the schema entries are incomplete.

The schema interpreters can either give access to the schema describing the entire database or to a subschema which is restricted to

information for which a given user is authorized.
We expect that schemas and subschemas will use similar representations so that a single schema interpreter can serve in either environment.

Examples of schema intepreters exist in all databases. Abrial, et al, (1970) has documented the schema interpreter for Socrate in detail. Extendible schema concepts are being developed at VisiCorp for their VISI ON (TM) system.

M6. Subschema generator.
An optional module associated with a schema based system generates subschemas defining views for specific users. A subschema generator uses as input the main schema of the database and generates a subschema which has the appropriate restrictions and limitations for a specific user.

Subschemas may also be created to operate on specific nodes of a distributed network. A node-based subschema will use communication functions in order to retrieve information from other nodes. Information obtained from remote nodes may be cached locally in order to enhance system performance. If operations using cached information are executed, a verification stamp is transmitted to assure that the cache entry was still valid.

Interpretive subschemas are used in IBM System R and consist of data manipulation satements which dynamically reformat the database in order to provide ghe required view. In CODASYL systems, subschemas are subsets of the logical schema specification, and when included during the compilation process of database accessing programs, cause references to database variables excluded from the subschema to be ignored. In IBM IMS special tables (PCB's) are compiled which limit programmed access to excluded record segments at execution time.

3.1.3 User accessible modules.

The externally accesible modules pose the most stringent specification needs, since here incomplete specifications can lead to problems in database portability.

M7. A database management language (DML) interpreter.
The DML interpreter is one of the alternative modules which executes instructions provided by the ADA programmer. Statements and parameters given in the DML language are transmitted in a formal way

to the interpreter. The interpreter for retrieval:

1. uses a schema in order to locate the data and define an access path
2. executes instructions to the file access system in order to retrieve the information
3. transform any representation differences
4. return the results to the user.

Similar steps are executed for data insertion and for data update. Database management language commands also include statements which define the beginning and the end of transactions and which define integrity requirements of the transaction.

Transaction begin and commit statements will also cause actions to be carried out by the backup module.
Integrity control statements will cause actions to be carried out by the access locking module.

Database manipulation statements are central to every database system. Language proposals which are intended to be applicable to a wide variety of implementations have been published by Date. An overview of database statements for a relational algebra is found in the MIT MACAIMS system documentation. Honeywell's Multics MRDS appears to be a successor to that development. Both System R and INGRES from UC Berkeley reports include description of the management of relational calculus statements.

M8. Database manipulation language compiler.
For routine programs which require a higher level of performance, a preprocessing database manipulation language compiler provides an alternative to the interpreter. A program which includes database manipulation statements to be compiled should be indistinguishable from a program where the statements are used to drive the interpreter.

An ADA program which contains database manipulation statements is transformed by the compiler into an ADA program which is expanded to directly call the lower level subsystem routines which execute the statements. An important aspect of this transformation is that now schema information is merged at compile time and does not have to be accessed during transaction execution time.

The obvious disadvantage is of course that programs, once they have been compiled, will have to be recompiled in response to schema changes. An adequate programming management environment is required to assure synchronization of schema changes with the data manipulation programs that have been compiled.

Preprocessing programs exist for most CODASYL implementations in COBOL and PL/1 languages, for INTEL/MRI System 2000 in COBOL, PL/1, and FORTRAN, and for many other systems. The query language in System R is compiled at first use, and automatically recompiled when schema changes obsolete the compiled version. Optimization of relational algebra sequences is performed in IBM Great Britain's PRTV system and well documented in the reports issued there. Methods for optimization of programs which access the database are described by Finkelstein from Stanford and IBM San Jose research.

M9. Query language.
In order to provide convenient direct access to a database management system a query language is essential. These are the languages oriented towards flexible request specification on an on-line terminal and provide support for interactive decision making.

Query languages can come in many flavors from simple interrogative routines to natural language processing routines. All of these approaches will depend heavily on semantics stored in the schema. Interpretation of the requests is the dominant method for handling query languages.

In a modular system the query processor will translate the queries first to database manipulation statements. These may then be interpreted in turn, or, at times, compiled and immediatly executed. Typically each query will be regarded as a single transaction and cause transaction begin and commit statement to be emitted as well.

The database itself may be accessed in order to correctly understand the query. We will describe some typical query language below, but do not expect that ADA specifications will be needed at this point for the external interfaces of these modules.

Interrogative approaches.
An interrogative query system will create and display menus based on the contents of the schema from which the user can select the required data elements. Once a subset of the database is identified

key values from the database will be presented for selection. Such an approach can also present the data in tabular form on the screen and permit updates by the execution of screen editing statements by the user.

A powerful example of this style of access is IBM's Query-by-Example approach. Similar approaches are found in many office systems.

Command style languages.
Many current systems provide direct access to the database using commands closely modeled on the set of available access statements. In order to have adequate power these languages must also provide the ability to collect intermediate results into temporary workspaces. These languages are typically used by specialists. The interactions may be noted in scripts in order to make repetitive usage more convenienent.

The EASYTRIEVE product is an example of a comprehensive language of this type.

Relational languages.
A relational query interface expects the user to know the layout of the database in terms of relations and attributes. Many of the languages are based on the relational calculus. The explicit use of workspaces can often be avoided. Correctly phrased queries may be compiled into optimal sequences of data manipulation statements. The scripts may be kept and perhaps edited for reuse.

SQL of IBM and SEQUEL of INGRES provide such facilities.

Natural languages.
A natural language interface would have a vocabulary which is comprised out of some operational verbs, the terms in the schema, and the lexical terms (the key strings), from the database itself. Such a natural language query system could be easily ported from application to application.

Research at the AI center of SRI international has developed tools to build such interfaces (IDA, LADDER, CLAUS), and the ROBOT system from AIC has been made available to access the CODASYL style IDMS system.

3.1.4 Internal database modules.

The next set of modules are used for internal database support.

M10. Access locking.
In order to assure consistency of responses and integrity of the database during multiple write operations an archiving module will be invoked from the other modules at critical junctions. We expect that users can decide to operate with different levels of consistency.

- o Queries might be of a type requiring complete lockout of all users in order to assure a consistent and unchanging database during such queries. Such locking will satisfy the strictest audit requirements, but severly affect the response of a shared database for all others.

- o Multiple queries may proceed in parallel as safely as with full lockout if transactions do not interfere with each other. A locking module to support this type of protection has to note for each transaction the read and write requests, and avoid conflicts. Three strategies are available:
  1. all requests are prespecified, and those which may conflict are delayed until they are safe.
  2. all requests are pre-executed. When a definite execution is requested any intervening access is noted, and, if there was an interfering interaction, the transaction is forced back into a pre-execution phase.
  3. the progress of the transaction is monitored, and if the access module recognizes a potential conflict it can cause a delay or an abort of the request.

- o Relative consistency can be assured by using a consistent set of after-images based on the time point that a query was asked. This permits concurrent operations to continue with the realization that current changes may occur in the database which will not be reflected in the answer being obtained by this query. To support this approach the access locking module maintains a table of record references matched to buffer references keyed to a user identification.

All these methods are available. Many simple systems use primitive lockout approaches, restricting either greatly the flexibility of transactions or disabling large sections of the database. Systems using transaction-abort schemes are found in systems which already have the capbility to recover from errors through logging and recovery schemes. Honeywell IDS II provides comprehensive facilities of this type. Research at CCA is developing the third approach using parallel access to older buffers.

M11. Scheduler.
The scheduler has as task to manage the interactions of multiple requests, multiple users and processes, and multiple devices which can serve the database in parallel. A scheduler is also closely related to the access locking module because a scheduler is not free to rearrange operations into an optimal sequence when this would conflict with consistency requirements recognized by the access module.

The scheduler also has to operate closely with the operating system and would probably be the module most affected in the move of a database management system from one operating system environment to another.

The objective of scheduling in transaction management for databases is different from the objective seen in typical timesharing systems. In timesharing systems, the objective is to give users equitable interactive access to the computer's resources. In a transaction system the data are considered to be the scarce resource and, since data cannot be used while another user has locked them, the objective is to give highest priority to the user who currently holds the most resources or who potentially can release held resources most rapidly.

Letting transaction schedulers operate within a generalized programming environment can lead to conflicts which can disturb the operation of other users which operate in a timesharing mode and the database management itself.

Scheduling for database management systems is not as well understood as scheduling for timesharing systems. Considerable experience has been gained in computer systems which are optimized for transaction handling. Examples of such systems are the Tandem Computer Systems and perhaps the CICS Systems on IBM commercial computers.

M12. Reorganization module.
As a database is affected by updates over time the intended locality of data, on which much of its performance characteristics may depend, can be seriously disturbed. Reorganization is akin to garbage collection, except that degradation due to a poor storage allocation tends to affect performance, rather than block computation entirely.

Reorganization should proceed without the user being aware of it. This is often accomplished by chosing odd times for reorganization. This solution is not adequate in a general sense. To avoid a negative performance impact when reorganization is done at active times, it is desirable that it can be performed for small parts of the database at a time.

Reorganization has a potential to invalidate cross references within the database. If cross references are handled indirectly the reorganization module may be involved automatically or manually. During its operation it may have to temporarily lock portions of the file access systems and hence restrict user access.

M13. Distributed query support.
In order to support queries to remote computers, an application level protocol and its supporting programs have to be provided.
We envisage that such a module will remotely execute instructions which are similar to database management language instructions.
If data paths are frequently transversed it may need to create temporary replicated files using the file access system as a cache on the local computer.

Experiments involving distributed query processing are being performed at the XEROX PARC lab and at IBM San Jose Research.

3.2 Specific functional capabilities of operational systems.

When we consider operational systems we concentrate still on functions related to the types of modules that we have presented in Section 3.1, and their interfaces. It is not clear to what extent these modules today could be copied and integrated into a successful DBMS.
We indicate with each system the organization and source language for the implementation. This list could be greatly extended, appendix B provides a basis.

3.2.1 Separation of Storage and Access System Interface

System-R (IBM {PL/1})
System-R has in RDS a subsystem which provides a formal interface between a simple storage mechanism and a relatively simple record management system. The original facilities were of RDS were based on the experimental Cambridge Monitor System developed many years ago.

The separation of function, which was created this way, has made it more convenient to implement System-R on distinct IBM operating systems as VMS, MVS, and the smaller operating systems.

FLASH (Stanford CSD (PASCAL)).
The FLASH implementation of a B-tree based file access system has identified formally the parameters that are required to move a file access system from operating system to operating system.
Experimental versions of FLASH have been implemented for DEC-20, UNIX, and IBM computers and are serving as models for microprocessor implementations.

3.2.2 Separation of access structure and data storage.

ADABAS (software AG,(assembler))
ADABAS has access structures which can be defined in the schema to be maintained continuously or can be defined within a query to be created dynamically if needed. The description of these features is unfortunately very poor, so tht that these facilities are not easily used by programmers.

SYSTEM 2000 (Intel-MRI (assembler- some FORTRAN))
All access information for the hierarchical structure is kept on distinct files.

3.2.3 Database performance.

IMS (IBM 370 series (assembler, PL/S))
Systems which employ refernces among records and control clustering can provide very high transaction performance, especially where the usage paatern is known. For instance, TRW is moving its credit inquiry system which services about 350,000 transactions per day to an IMS environment.

3.2.4 Subschema management.

IMS (IBM 370 series (assembler, PL/S))
IMS permits definition of logical subschemas which provide completely different view than the actual database structure implies.
These views are available to users under the name "Logical Databases". All logical databases and the physical database itself is restricted to a hierarchical structure. However, the hierarchical structures do

not have to overlap.

SQL/DS (IBM DP)
Subschemas can also be define simiar to query statements, and be interposed automatically at query execution time.

3.2.5 Access to heterogeneous databases over a network.

Multibase (CCA)
The work on multibase demonstrates that front ends can be built that link heterogeneous databases together and hence provide one consistent high level model for distinct implementations.

3.2.6 Natural language query systems.

IDA (SRI international ¼Interlisp½)
Query systems developed in research environments as LADDER and CLAUS have shown that natural language access for databases is feasible.

ROBOT (Artificial Intelligence Corporation ¼MAClisp½)
A commercial front-end natural language system, ROBOT, is now being marketed as a frontend to a network database system (Cullinane IDMS).

3.2.7 Logical to physical mapping.

ORACLE (Oracle Systems)
A number of relational front-ends are now being provided for nonrelational implementations. The most important example is ORACLE which uses an internal hierarchical structure.

RIDMS (Cullinet {COBOL?})
Relational frontend processors also are being provided for Cullinane IDMS, the major network database for large IBM computers, and for MDBS a network structured system operating on microcomputers.

3.2.8 The ability to combine compiled data manipulation programs.

IDS II (Honeywell {COBOL})
Preprocessors to compile data manipulation programs and make them independent of the schema at execution time exist for most CODASYL implementations.
The earliest example of such processors are the IDS systems and a

recent example re the SQL/DS preprocessors for COBOL and PL/1.
The latter include facilities for automatic recompilation when
the compile modules are invalidated due to database reorganization.

3.2.9 Schema interpreters

INGRES (RTI (C))
Interpreters which uses schema to direct the translation of
data manipulation to languages are in common use.
They are part of the INGRES system (UC Berkeley), TOD (Stanford), etc.

3.2.10 Scheduler.

CICS (IBM).
An example of the transaction scheduler operating in a general
programing environment is CICS (IBM). CICS schedules transaction
execution with as objective the minimization of the time taken for
indiviual transactions and thus minimize resource allocation
requirements. The major constraint on CICS operations is the buffer
allocation required to keep multiple transactions active.

3.2.11 Recovery logging.

IMS (IBM 370 series (assembler, PL/S))
The IMS system has demonstrated that long term reliable operation of
databases is feasible while extremely high transaction rates are
being supported. The complexity of the system is unfortunately such
that training of support programmers is difficult and simple
applications are discouraged.

3.3 Operational Databases.

Several hundred database management systems of widely varying
capabilities are now on the commercial market. For instance, on
microcomputers, the use of database management systems follows in
frequency the use of spread sheet packages and word processors,
at least if games are excluded.

There are several database management systems available for every
major commercial computer currently being manufactured.
We refer to an appendix taken from Appendix B of Wiederhold: Database
Design, 2nd edition, McGraw-Hill for a listing of database management systems.

Case Studies

In this section we will mention some systems which warrant a deeper analysis. The reason for selecting them are their practical importance now or their expected relevance in the future.

3.3.1 SQL/DS (IBM System Development Division)

Development:
This system is the first of a family of relational systems from IBM. The design is largely based on System R developed at IBM Research in San Jose. The manager of the project doing much of its development was J.F. King. I believe that the current manager is Bob Taylor.

Description of system:
SQL/DS is a commercial derivative of a developmental project at IBM Research. It is a complete database management system and includes a storage management system which uses operating system facilities, a query processor which includes commands for transaction management, facilities for system backup, preprocessors for PL/1 and COBOL programs which include SQL/DS statements.

Those statements are not otherwise integrated into the PL/1 or COBOL programming languages. The SQL/DS statements define result relations. The linkage to the programs is via cursors and shared area to hold the curent record.

Views may be prespecified using SQL/DS statements. The views and the results from SQL/DS queries are interpreted at query processing time. The code from an interpretation is saved together with the actual query to permit reuse or reinterpretation as needed.
The results of a view or query are not neccessarily materialized in a working file. Commands to move the cursor can cause further database processing as needed.

Performance and Quality:
No formal specification on performance are currently being provided. A high degree of optimization of relational queries is part of the query processing program. However cross references between relations are always symbolic and all such references are resolved at query time. Other linkage types have been discussed, but were never implemented. We do not believe that the performance of SQL/DS is such that it is suitable for very large data-processing type applications.

SQL/DS at this state is a commercial product and we assume that it has a high degree of reliability.

Development resources:
I can only guess at the size of the development team. It appears that it was modest, less than ten people. Successor products are still being announced and developed.

The development environment for SQL/DS included using standard IBM operating system facilities and the PL/S system implementation language. The existence of System R provided a very convincing specification of the system to be implemented.

Use:
SQL/DS has been released since early 1982 and I do not know how many systems have been installed and to what extent it is being used.

3.3.2 IDMS (B.F. Goodrich Company, Cullinet)

Development:
IDMS was originally developed for commercial support within B.F. Goodrich Company. Subsequently the system was sold to and marketed by Cullinane Corporation which renamed itself Cullinet Corporation after adding facilities for distributed databases to IDMS.

Description of System:
IDMS is an implmentation of the CODASYL 1972 specifications with some extensions to enhance the distinction between logical and physical schema specifications. Options provided with IDMS include a natural language frontend, ROBOT, provided by Artificial Intelligence Corporation, and recently a relational language processor.

Performance:
IDMS transaction programs can use specified linkages between relations and if written to make good use of these linkages can provide very high performance. Initial entry points into the database are found by using hashing techniques. The processing of more general queries is obviously not quite symmetric.

The performance will depend greatly on whether prespecified links can be followed. The schema permits specifiction of locality directives and device alloction.

Use:
IDMS has been in operation for more than ten years now.
It consistently gets very high marks in terms of product quality and

reliability. It gets lower marks in terms of flexibility.

Development resources:
A large fraction of Cullinet Corporation is devoted to the development maintenance and marketing of IDMS. However, most of the current efforts are in the development of value-added products, query processors, auditing packages, report generators, and programs for selected industries.

3.3.3 ORACLE (Oracle SystemsInc.)

Development:
Oracle was developed originally under U.S. Government Contract, and subsequently commercialized and marketed by Relational Software, Incorporated, in Menlo Park. This company has been recently renamed Oracle Systems.

Description:
Oracle is a relational databases and uses the same query language used by IBM's SQL/DS. It includes additional data types, type and date, and this being modified to work in a distributed system environment.

Oracle's design uses a hierarchical structure so that for the subset of queries which follow the hierarchy the performance can be very high. Oracle provides views for users which are limited to access a subset of the database.

Development resources:
Approximately 20 people are now involved in further development and technical customer service for Oracle.

Use:
Oracle is operational on DEC PDP-11 and VAX computers. Implementation for the IBM 360's types are due to be available. Oracle has been available since 1979. Oracle has been used in a number of installations with a fair amount of success although it is not as mature as some of the other systems mentioned.

4. A DEVELOPMENT PLAN AND AN ANLYSIS OF ITS SUITABILITY.

As discussed earlier, existing database management systems are highly integrated and the subsystems are not easily severable.
This means that the choice for an ADA environment is to either

1. Select some best database management system and encapsulate the entire system within ADA

2. Develop the specifications for database support within ADA and, in order to provide support for these specifications, implement a set of simple modules which define the component functions of a database management system.

In this report we favor the latter approach.

We believe that the inclusion of an existing database management system with an ADA environment will restrict flexibility. We also believe that the the development of an adequate interface, required for the first approch will cost nearly as much to provide as as the programming of a set of simple modules. Once a set of simple modules has been defined and implemented they can provide a basis for commercial development of improved systems using replacement modules.

It is highly unlikely that any capsulated database will use file structures which are compatible with the file structures to be supported by file system packages under ADA. Unless a decision could be made to adopt a portable database management system which can operate identically on the range of machines which we expect ADA to support the encapsulation approach will be very limited.

4.1 Specification development

The critical portion of a new development path is the definition of appropriate specifications. In order to permit the required growth the most critical specification is that of the schema manipulation module.

It has to be possible to add semantic descriptions to the schema nearly ad infinitum. Any descriptions which are not used by modules can be ignored but the meta-data in the schema provides the formal communication basis for all the cooperating modules. We would like to see in this sense a capability where additions to the schema can be handled in a manner which is synchronous to the addition of types in an application program.

4.2 Sample module implementation.

When the specifications are in reasonable shape sample modules. can be developed. Good sources would of course be institutions which have developed modules in the specific areas, although an understanding of ADA and its objectives will also be needed.

The encapsulation of the modules will be severly tested if implementation is distributed.

4.3 Schedule.

Without a detailed analysis it seems that specifications in the well-understood data-processing areas can be completed in less than a year, and that another year would provide functional modules.

In the database management area two years for the specifications and a year for module development and year for integration seems possible, perhaps optimistic. A deadline of 1989 seems very schievable however.

5.0 ACKNOWLEDGEMENT

Work on advanced file systems and databases related to these ideas was performed under sponsorship of NIH Division of Research Resources from 1965 to 1975. Some notions of modularization were developed in conjunction with the 1974 IFIP TC-4 meeting in Cargese, France. They were expanded and reconsidered at a Honeywell International Workshop on Database Management, Spring Hill Center, Oct. 1980.

Current work in the KBMS project is supported by ARPA through ONR and NAVALEX, contract N39-82-C-250. Many students have contributed to this research. Arthur Keller reviewed the draft and is the author of Appendix C. Jayne Pickering did the original transcription.

6.0 REFERENCES.

References are not included in this report. The various companies cited should be contacted for recent manuals, and not all the relevant information is in the open literature. Many references are included in Wiederhold, Database Design, 2nd ed., McGraw-Hill 1983, and a voluminous bibliography (about 3000 annotated entries) can be provided on request, or shipped over the ARPAnet.

Appendices

Appendix A : CODASYL 1978 Logical and physical schema specifications.

Appendix B : Database Management Systems (from Wiederhold: Database Design).

Appendix C : Arthur Keller: Indexed File Access for ADA.

Appendix D : A sketch for a candidate database modularization
from Wiederhold: Futures in Database Management,
Honeywell International Database Workshop 1980.

```
13-Sep-83 10:34:27-PDT,297;000000000000
Date: Tue 13 Sep 83 10:34:27-PDT
From: Gio ¼Wiederhold@SRI-AI.ARPA½
Subject: Re: DOD Report
To: ARK@SU-AI.ARPA
In-Reply-To: Message from "Arthur Keller ¼ARK@SU-SCORE.ARPA½" of Sun 11 Sep 83 16:56:18-P

thanks.
 I711 include your paper, app B from the book and a fig from Chap.8
-------
13-Sep-83 17:24:55-PDT,1563;000000000001
Return-Path: ¼WIEDERHOLD@SUMEX-AIM.ARPA½
Received: from SUMEX-AIM.ARPA by SRI-AI.ARPA with TCP; Tue 13 Sep 83 17:24:49-PDT
Date: Tue 13 Sep 83 17:24:54-PDT
From: Gio Wiederhold ¼WIEDERHOLD@SUMEX-AIM.ARPA½
Subject: Gio Wiederhold ¼WIEDERHOLD@SUMEX-AIM.ARPA½: Mark Linton ¼linton@Shasta½: ref  en

To: wiederhold@SRI-AI.ARPA

mov
                ---------------

Mail-From: WIEDERHOLD created at  9-Sep-83 15:46:34
Date: Fri 9 Sep 83 15:46:34-PDT
From: Gio Wiederhold ¼WIEDERHOLD@SUMEX-AIM.ARPA½
Subject: Mark Linton ¼linton@Shasta½: references
To: wiederhold@SUMEX-AIM.ARPA

Received: from Shasta by SUMEX-AIM with Pup; Fri 9 Sep 83 15:37:31-PDT
Date: Fri, 9 Sep 83 15:42 PDT
From: Mark Linton ¼linton@Shasta½
Subject: references
To: WIEDERHOLD@SUMEX-AIM.ARPA

FYI:

Here are three references about my programming environment - database stuff.
Thought it might come in handy if the subject should come up in your travels.
Each is co-authored with my advisor, Mike Powell.

    "A Database Model of Debugging", Proceedings of the ACM SIGSOFT-SIGPLAN
    Symposium on High-Level Debugging, in SIGPLAN Notices, Vol. 18, No. 8,
    August 1983.
```

"Visual Abstraction in an Interactive Programming Environment", Proceedings of the SIGPLAN '83: Symposium on Programming Language Issues in Software Systems, in SIGPLAN Notices, Vol. 18, No. 7, July 1983.

"Database Support for Programming Environments", Proceedings of the Database Week Special Session on Engineering Design, May 1983.

```
SCHEMA NAME IS model_schema_name.
   /* Within a SCHEMA several areas can be defined */
AREA NAME IS area_name.
   /* and several record types may live within an AREA  */
RECORD NAME IS record_type_name     /* implements a relation table */
     WITHIN  { ANY AREA                          }
             { area_name                         }
             { AREA OF OWNER OF set_name         }
    [ KEY key_name IS [{ASCENDING / DESCENDING}] data_in_key [, ... ]  ]
    [   DUPLICATES ARE{FIRST / LAST / NOT ALLOWED / SYSTEM DEFAULT}     ]
    [   FREQUENCY OF  [DIRECT][SEQUENTIAL]  RETRIEVAL IS HIGH           ].
   /* and then the attributes or data elements are specified  */
level_no  data_name
          [PICTURE . . /* a COBOL style format specification  */ ]
          [TYPE   { {BINARY } {FIXED} {REAL   }                             } ]
          [   IS  { {DECIMAL} {FLOAT} {COMPLEX} number_size [.frac_size]    } ]
                  { {BIT / CHARACTER}  size [DEPENDING ON variable]         }
                  {  implementor_name  /*  for special types             */ }
          [OCCURS{integer_count / variable_count} TIMES]
          [CONVERSION IS NOT ALLOWED]
           CHECK IS [ NONULL                                ]
                    [ PROCEDURE procedure_name              ]
                    [ VALUE [NOT] literal_1 [THRU literal_2] ]
   /* Derived data  */
          [SOURCE IS some_data_identifier OF OWNER of set_name_1]
          [RESULT OF PROCEDURE derive_procedure ON CHANGE TO                      ]
          [ {ALL DATA              } {OF THIS RECORD                            } ]
          [ {DATA identifier [, ...]} {OF ALL MEMBERS           } OF set_       } ]
          [ {TENANCY               } {OF MEMBER record_name_m } _name_m        } ]
        /* CHANGE OF TENANCY means change of link-set membership or ownership */
     /* The connections to be implemented in a schema are defined as follows */
SET NAME IS link_set_name         /* implements a connection */
    OWNER IS{record_name_o / SYSTEM }
    ORDER {PERMANENT} INSERTION {FIRST / LAST / NEXT / PRIOR / SYSTEM DEFAULT}
      IS  {TEMPORARY}     IS    {SORTED    {WITHIN RECORD-TYPE    }          }
                                {          {BY DEFINED KEYS [ ... ]}         }
    MEMBER IS record_name_m
      [DUPLICATES ARE NOT ALLOWED FOR attribute_k1 [, ... ] ]
        STRUCTURAL CONSTRAINT IS variable_a EQUAL TO variable_b [, ... ]
/* Omitted is detail of RESULT and SET. Also omitted are some size parameters, access
procedures locks, escape calls, as well as some FREQUENCY clauses for optimization advice
SET features relevant to data manipulation are shown in Fig 9-12 */
```

Figure 8-9 Data Description Language defined by the CODASYL DDL Committee (1978). Clauses in [ ] are optional, in { } are alternatives, with ... are repeatable.

```
STORAGE SCHEMA NAME IS storage_schema_name
   FOR schema_name SCHEMA
     [ REPRESENT  { ALL [EXCEPT] }  schema_record_name RECORDS  ]
     |            { ONLY         }                              |
     | AND        { ALL [EXCEPT] }  schema_linkset_name SETS    |
     [            { ONLY         }                              ]
     [ MAPPING FOR schema_record_name_y                              ]
     |      [ If condition ] STORAGE RECORD IS storage_record_name_x | .
     [                                                               ]
     STORAGE AREA NAME IS storage_area_name
        INITIAL SIZE IS integer_1 PAGES
        [EXPANDABLE [BY integer_2 PAGES] [TO integer_3 PAGES]]
        PAGE SIZE IS integer_4 { CHARACTERS / WORDS } .
      /* The clauses below are repeated for each storage record type */
STORAGE RECORD NAME IS storage_record_name_1
      [LINK TO storage_record_name_2 [ IS { DIRECT   } ]] , ... [...]
      |                              |    { INDIRECT } |
      [RESERVE integer_5 POINTERS]   [                 ]
      [ [ If condition ] DENSITY IS n_block STORAGE RECORDS PER b_train PAGES ]
      PLACEMENT IS
         { CALC [hash_procedure_name] USING identifier_1, ...                  }
         { CLUSTERED VIA SET schema_set_name                                   }
         {                     [NEAR OWNER storage_record_name_0]              }
         {                     [WITH storage_record_name_3]                    }
         { SEQUENTIAL { ASCENDING / DESCENDING } identifier_2, ...             }
         WITHIN storage_area_name [FROM PAGE int_8 THRU int_9] .
     /* And now come the actual field definitions */
level_no data_name
            [ALIGNMENT IS integer_10 {BITS / CHARACTER / WORDS}]
            [EVALUATION IS ON { ACCESS [STORAGE [NOT] REQUIRED] }
                              { UPDATE                          }
            [FORMAT IS /*  a variety of standard or implementor defined types */]
            [NULL IS {literal_value / COMPACTED}]
            [SIZE IS integer_size {BITS / CHARACTER / WORDS}] .
     /* The clauses below are repeated for each link_set connection in the schema */
SET schema_set    [ALLOCATION IS {STATIC / DYNAMIC }]
    POINTER { INDEX index_name                                  }
       FOR  { ... RECORD schema_record IS ... TO storage_record ... } .
```

These clauses relate the model CODASYL *definition of Fig. 8-9 to implementation concepts presented in Chaps. 2 to 5; the use of many of these clauses is described in Sec. 9-5-4. Omitted are* ACCESS CONTROL, *details of If* CONDITION, DENSITY, *and data alignment. Details of* SET ... POINTER *are given in Fig. 9-15.*

Figure 8-10 The 1978 CODASYL Data Storage Description Language proposal
Clauses in [ ] are optional, in { ... / ... } are alternatives, with [...] are repeatable

Database Manipulation The statement types to be available for manipulation of a 1978 CODASYL database are given in Table 9-4. Specific formats for the COBOL language are given in the documentation for COBOL of the CODASYL committee, and FORTRAN versions have also been specified

Each executable statement has a numeric code. The number is used when the manipulation functions of a DBMS are invoked by one of the different host languages.

Table 9-4 Declarations and Manipulation Commands for a CODASYL Database

/ Obtain access to the relevant portions of a schema its storage schema, and the contents of the database defined by it by invoking a subschema using* COBOL */		
DB	**sub_schema_name** WITHIN **schema_name** [; ACCESS CONTROL KEY = xxxx]	
LD	**keep_list_name** LIMIT IS **integer**	/* *to keep currency indicators* */
/ Transaction control */*		
13	READY	lock areas as specified in the **AREA** clause of the schema
01	COMMIT	release record locks and reset all currency indicators and keep_lists. The statements coded 02, 03, 04, 11, 12, 15, and 16 lock the records and link-set entries accessed.
06	FINISH	release locked areas
09	IF .	test database status and error condition codes
14	ROLLBACK	remove all database changes since **READY** or **COMMIT**.
/ Finding and manipulating records */*		
05	FIND	locate a record.
08	GET	obtain specified data items or all of the current record.
15	STORE	insert a record according to the schema specifications.
11	MODIFY	update the current record.
04	ERASE	delete the current record.
10	KEEP db_cp	obtain a currency indicator and place it into a **keep_list**.
/ Manipulation of link-sets */*		
02	CONNECT	establish **MANUAL** link-set membership.
03	DISCONNECT	remove a record from link-set membership.
16	RECONNECT	move a record from one link-set to another link-set.
/ Other */*		
07	FREE db_cp	release currency indicator, including any entries kept for currency indicator **db_cp** in a **keep_list**.
12	ORDER	sort the members of the current set logically so that they can be retrieved in a certain order.
	USE . .	declaration to identify procedure to be executed when an exception or error condition occurs and to identify access control procedures.

9-2-1 A Relational Calculus System

We will begin by presenting aspects of a language, SQL, used by several implementations of the relational calculus, and will then discuss some features of similar systems as well as some implementation issues. We focus on retrieval and present the principal command in Table 9-1, followed by examples of its use.

The SELECT command presents a result table based on attributes from the database tables listed in the FROM clause. The WHERE clause restricts the result to rows meeting certain conditions. Aggregation during selection and other commands will be touched on later. The notation follows Fig. 8-10.

Table 9-1 The Retrieval Command of SQL/DS

```
SELECT {            *              }                         /* * means all attributes */
       { attribute_ac [ , ... ]    }
   FROM table_name [tuple_variable] [ , ... [ ... ] ]
    [ WHERE selection_expression ]
    [ GROUP BY attribute [HAVING selection_expression], ... ]
    [ ORDER BY attribute [DESCending], ... ]
/* The principal primitive component of the Select command is: */
     attribute        /* must be listed in the schema of some FROM table */
     attribute_ac     /* is a simple attribute or
                         an arithmetic expression of attributes and constants. */
     /* tuple_variables are presented in Sec. 9-2-2. */
 Selection_expression ::=      /* a boolean expression using attribute_expressions: */
     [NOT] selection_expression [ [NOT] { AND / OR } selection_expression ]
     ( selection_expression )
     attribute = USER               /* user id, good for checking a VIEW  */
     attribute_expression
 Attribute_expression ::=      /* a simple or a complex conditional expression: */
     attribute_ac ⊜ constant
              /* ⊜ is one of the set { =  -=  >  >=  <  <= } */
     attribute_ac ⊜ attribute_ac
     attribute_ac BETWEEN low_attribute_ac AND high_attribute_ac
     attribute_ac [NOT] IN (constant_1, ..., constant_n)
     attribute ⊜ {ANY / ALL} (constant_1, ..., constant_n)
     attribute IS [NOT] NULL
     attribute [NOT] LIKE 'search_string'      /* See note in Table 9-3 */
/* Attribute_expressions can include other SELECT substatements: */
   /* If the subquery leads to a single value: */
     attribute_ac ⊜ ( SELECT ... FROM ... ... )
   /* If the subquery can lead to a set of values: */
     attribute_ac [NOT] IN ( SELECT ... FROM ... ... )
     attribute_ac ⊜ { ANY / ALL } ( SELECT ... FROM ... ... )
   /* If the result of the subquery is to be quantified (see Sec. 9-2-3) */
     [NOT] EXISTS ( SELECT ... FROM ... ... )
```

Database Systems

The systems in this list were chosen because of their ubiquity, their historical interest, their potential for experimentation, or their significance for further study. Other sources for references to database and file systems can be obtained from commercial software catalogs (DATAPRO, Auerbach, ICP Quarterly) or from surveys in computer magazines, for example, Krass[81]. CODASYL[71A] contains a detailed review of GIS, MARKIV, NIPS FFS, TDMS, UL 1, COBOL, DBIG, IDS, IMS, and SC-1. Martin in Nance[75] surveyed STAIRS, DIALOG, DATA CENTRAL, ORBIT used for MEDLINE, BASIS, SPIRES, LEADER, RECON, RIQS, INTREX, and NASIS. Kim[79] surveyed relational DBMS' and Brodie[82] includes a survey of relational systems: IDM, INGRES, MRDS, MRS, NOMAD, ORACLE, PASCAL R, PRTV, RAPPORT, SYSTEM R, QBE, RAPID, and cites a total of 60. Wiederhold[82] includes descriptions of distributed DBMS efforts. CODASYL[76] provides guidelines for system selection. Landau[76] produces a listing of on-line databases.

The commercial systems included below vary in price from several hundred dollars total to several thousand dollars per month.

Name	Year	Developer location	Computer	Type and features
ACCENT R	1981	National Information Syst Cupertino CA	DEC10/20	Com DBMS slc rel stq sch
ADABAS	1971	software ag Darmstadt FRG	IBM360/370 Siemens	Com DBMS hlc stq sch drf cip pri rec Atre[80]
ADEPT	1969	System Dev Corp Santa Monica CA	IBM360-50	Exp DBMS slc nlq(CONVERSE) pri Weissman[69]
ADMINS	1966	MIT&ADMINS inc Cambridge MA	DEC 11, VAX	DBMS slc sch rel trf McIntosh[68]
ALPHA	1971	IBM Research San Jose CA		Pro DBL slc rel Codd in Codd[71]

Name	Year	Developer location	Computer	Type and features
AMBASE	1979	Amcor Computer Corp Louisville KY	DEC 11 (RSTS)	Com DBMS sch isf
AMIGOS	1970	Comress Inc Rockville MD	IBM360/370	Com FMS hlc isf
APPEL IV	1974	SIS Paris France	IBM360/370 Burr 500/700	Com DBMS hlc hie drf isf
ASI INQ	1975	Applications Softw Torrance CA	IBM360/370	Com QUS for DL/1 slc hie stq
ASI ST	1969	Applications Softw. Torrance CA	IBM360/370 Univac70	Com QUS slc rpg tbq sch *for* sqf. isf IMS. TOTAL
Ass.PL	1967	General Motors Warren MI	IBM360-67	Inst DBMS hlp(PL 1) rnf gra Dodd[66]
AUTONOTE	1969	Univ of Michigan Ann Arbor MI	IBM360/370 (MTS)	Exp SATDBMS slc txt sch sqf drf Reitman[69]
BASIS	1970	Batelle Mem Labs Columbus OH	CDC6400 DEC 10/20.VAX IBM 370 UNIVAC1100	Inst IRS stq bib rpg Fried in Walker[71]
BEAST	1968	Brookings Inst Washington DC	DEC PDP10	Inst SATDBMS slc sch sqf Kidd[69]
BIS	1967	Am Tel &Tel New York NY	IBM360	Inst DBMS hlc hie sch Benner[67]
CAFS	1976	ICL Stevenage UK		Dev DBCMP rel Babb[79]
CASSM	1975	Univ of Florida Gainesville FL		Exp DBCMP Su[79] Hawthorn[82]
CDMS	1969	System Dev Corp Santa Monica CA	IBM360/370	Com DBMS service slc irq ixf *see* TDMS
CDMS	1974	Digital Eq Corp Maynard MA	DEC 11	Com DBMS slc hie trf *see* MUMPS
CFS	1980	Carnegie-Mellon U. Pittsburgh PA	DEC LSI-11s	Exp DFMS
CIA	1982	Computer Invest.Adv. Sewickley PA	Apple	Com DBMS 1-rel alg hlc(BASIC) rpg sch isf
COGENT	1969	Comp Sciences Corp Los Angeles CA	IBM7090. 370 Univac1100	Com DPG hlp(COBOL) sch hie isf ixf
CONVERSE	1967	System Dev Corp Santa Monica CA	IBM360-67 ANFSQ32	Exp QUS net nlq vrf Kellogg[68]. in SIGIR[71]

Name	Year	Developer location	Computer	Type and features
COSTAR	1978	Mass Gen Hospital Boston MA	DEC PDP15,11 Tandem	SATBMS slc trf(MUMPS) Barnett[79]
CREATE	1975	Complete Computer Sys. Horsham PA	DataGeneral	Com DPG ixf pri
CREATE/3000	1977	CRI Inc Mountain View CA	HP 3000	Com DBMS hlc rel stq ixf
CZAR	1970	Crown Zellerbach San Francisco CA	IBM360/370	Inst QUS hlc sch isf Palmer[75]
DATA ANALYZER	1971	Program Prod. Nanuet NY	IBM360/370	Com IRS slc rpg sch sqf isf
DATACATALOG	1974	Synergetics Bedford MA	IBM370 OS,DOS UNIVAC	Com DDICT slc rpg sch isf DMS1100 IDMS IMS S2000
DATACOM	1970	Aplied Data Res. Dallas TX	IBM360/370	Com DBMS hlc sch sqf ixf cpr
DATA COMPUTER	1971	Comp Corp of Am Cambridge MA	PDP10 (TENEX)	Dev DBMS hlc hie stq vrf *for* ARPAnet
DATAMAN	1975	Dataman Ltd Calgary Alberta	IBM360/370	Com FMS slc rpg sch sqf
DATAMANAGER	1976	MSP London UK & Lexington MA	IBM360/370	Com DDICT slc rpg sch ADABAS IMS MARKIV SYSTEM 2000 TOTAL
DATAMASTER	1980	Microsoft Seattle WA	Apple 8080	Com FMS rpg sch sqf
DATASAAB	1974	Saab-Scania AB Linkoping Sweden	SAAB D22/D23	Com DBMS hlp(COBOL) net sch rnf pri Bubenko[75]
dBASE II	1981	Ashton-Tate Culver City CA	Z-80 CP/M	Com FMS, Join stq rpg ixf(1 *updated*)
DBC	1978	Ohio State&UNIVAC Columbus OH		Dev DBCMP sch ixf Banerjee[79] Hawthorn[82]
DBMS	1977	Prime Computer Inc Wellesley Hills MA	Prime	Com DBMS hlp net sch stq(IQL) rnf rec pri
DBMS10/20	1973	Digital Eq Corp Marlboro MA	DEC 10/20	Com DBMS hlc net(1973) sch stq(IQL) rnf rec pri
DBMS11	1979	Digital Eq Corp Marlboro MA	DEC 11	Com DBMS hlc net(1973) sch rnf
DBMS990	1980	Texas Instruments Austin TX	Ti990	Com FMS hlc(PASCAL COBOL FORTRAN) hie-tbq isf

Name	Year	Developer location	Computer	Type and features
DBMS1900	1974	ICL London England	ICL 1903	Com DBMS hlc sch isf pri rec
DBOMP	1970	IBM White Plains NY	IBM360/370	Com BOMP net stq DL/1 files: CFMS *for* isf
DBS90	1972	Sperry Rand GmbH Frankfurt a/M FRG	Univac90	Com BOMP hlc(COBOL) net
DB/DC	1975	IBM White Plains NY	IBM360/370 IMS	Com DDICT slc sch rpg (CMIS *for manufacturing*)
DIALOG	1967	Lockheed Res Corp Palo Alto CA	IBM360	Com IRS *service* slc irq hie bib Walker[71]
DIRECT	1977	Northwestern Univ Evanston IL		Pro DBCMP DeWitt[79] Hawthorn[82]
DISAM	1975	Four Phase Systems Cupertino CA	4phase70	Com FS *for* DDBMS hlp(COBOL) isf ixf
DL/1	1968	IBM White Plains NY	IBM360/370	Com FMS hlc sch hie sqf isf drf stq(CICS)
DM1.5 *also* SC-1	1966	Auerbach Philadelphia PA	Univac418 IBM360/370	Com DBMS slc hie rpg sch ixf CODASYL[71A]
DMS1100, 90	1971	Univac Minneapolis MN	Univac1100 Univac90	Com DBMS hlp(COBOL) net(1969) rnf rec vie
DMS170	1977	Control Data Corp Minneapolis MN	Cyber170	Com DBMS hie sch sqf *derived from* (MARS) isf drf
DMS/1700	1975	Dedicated Systems Chicago IL	Burr 1700	Com FMS ixf
DMS II	1972	Burroughs Pasadena CA	B1700 *to* 7700	Com DBMS hlp(COBOL, ALGOL) net sch sqf isf ixf cpr rec
DMS IV *based on* IDS	1972	Honeywell Inf Sys Phoeniz AZ	H60	Com DBMS hlp(COBOL) net(1973) rpg rnf rec
DPL *also* IPL	1976	National Information Syst, Cupertino CA	DEC10/20	Com DBMS hlc(FORTRAN COBOL) sch isf pri
DYL250	1971	Dylakor Comp Syst Encino CA	IBM360/370	Com FMS slc rpg stq sqf isf
EDEN	1982	Univ of Washington Seattle WA	DEC VAX	Exp DFMS obj Jessop in Wiederhold[82]
EDMS	1969	Control Data Corp Brussels Belgium	CDC6400 Cyber	Com DBMS hlc(FORTRAN) stq sch(ANSII) isf Nijssen[77]

Name	Year	Developer location	Computer	Type and features
FACETS *was* PRISM	1981	Synergistics Bedford MA	IBM 370	Com DDICT slc rpg
FOCUS .		Information Builders New York 10001 NY	IBM 370 CMS,TSO	Com QUS irq rpg isf,DL/1,IDMS pri
FORDATA	1974	CSIRO Canberra Australia	CDC CYBER76	Inst DBMS hlp(FORTRAN) net sch
FORIMS	1970	Nippon Univac Tokyo Japan	Univac1100	Dev DBMS hlc(FORTRAN) net ixf rec
FORTE	1959	Burroughs Corp Paoli PA	B2500/3500 B1700-7700	Com FMS hlc(COBOL) sqf isf drf ixf rnf Chapin[69]
FRAMIS	1977	Lawrence Liv.Lab. Livermore CA	CDC 7600 CRAY DEC VAX(VMS)	Dev DBMS rel alg stq rnf(CODASYL)
GIM	1967	TRW Systems Redondo Beach CA	IBM7094,360/ 370 Univac1100 Honeywell6000	Com QUS slc stq sch rec drf rng Nelson[67]
GIS	1966	IBM White Plains NY	IBM360/370	Com QUS hlc(COBOL, PL/1) hie stq sch sqf isf
GMIS	1975	MIT Sloan School& IBM Cambridge MA	IBM370 (XRM)	Dev SATDBMS(Dec.supp.) rel vrf Donovan[76]
iDBP 86/440	1982	Intel-MRI Austin TX	*links to* IEEE 488, ETHERNET	Com DBCMP rel pri rec
HOPS	1975	Technicon Haifa Israel	Burroughs 126	Exp DBMS hie trf Reiter in Kerr[75]
IDM 500	1981	Britton-Lee Los Gatos CA	VAX *on* IEEE-488 *or* intlgnt term	Com DBCMP rel isf,ixf pri
IDMS	1972	Cullinane Corp Westwood MA	IBM360/370 ICL1902 Univac70/90 Siemens4004	Com DBMS hlp(COBOL) nlq(ROBOT) rpg (CULPRIT) net(1973→) sch DDICT drf rec
IDS I, II	1962	Honeywell Inf Sys Phoeniz AZ	H200, H60, H6000	Com DBMS hlp(COBOL) net(73) rnf rec Bachman[66]
IDP EDMS	1978	Honeywell *was* XDS Los Angeles 45 CA	H66 Sigma 6,7,9	Com DBMS hlc(COBOL) net rnf ixf sch rec
IFIP *also* RIM	1978	Boeing Computer Co (IPAD) Seattle WA	DEC VAX IBM	SATDBMS *for* CAD/CAM net(1978) hlp(FORTRAN)
IMAGE	1974	Hewlett-Packard Santa Clara CA	HP3000, HP2100	Com DBMS hlc sch pri stq net(2 level) drf, rnf)

Name	Year	Developer location	Computer	Type and features
IMARS	1971	Computeria Inc Braintree MA	DEC PDP10	Com QUS slc stq rpg rec
IMS-2/VS	1968	IBM White Plains NY	IBM360/370	Com DBMS hlc *multi*-hie sch(DL/1) stq rec(-VS)
INFOS	1975	Data General Southboro MA	DG Nova, Eclipse	Com FMS hlc hie isf ixf stq
INGRES	1973	Un.of CA&Relational Technology. Berkeley	DEC 11. VAX (UNIX. VMS)	Dev DBMS slc hlp(C. so.) rel stq gra pri Held[75]
INQUIRE	1969	Infodata Systems Falls Church VA	IBM360/370	Com IRS hlc rpg stq sch pri; Dev DDBMS(IQNET)
INTREX	1966	Mass Inst. Tech Cambridge MA	IBM7094, IBM360	Inst IRS irq/stq bib ixf Walker[71]
IS 1 *later* PRTV	1971	IBM UK Research Peterlee UK	IBM360/370	Dev DBMS hlp(PL/1) rel alg vie com stq Todd[76]
ISAM70	1974	Software70 Anaheim CA	Any FORTRAN system	Com FS hlc(FORTRAN) isf
LADDER	1977	SRI International Menlo Park CA	DEC PDP10	Exp IRS nlq net(DBMS20) Hendrix[78]
LEADER	1967	Lehigh Univ Bethlehem PA	CDC6400	Inst IRS irq bib ixf Hillman[69]
LEXICON	1976	Arthur Anderson Chicago IL	IBM360/370, System 3	Com DDICT slc IDMS IMS TOTAL
LEXIS	1978	Mead Data Central Lexis New York NY	IBM370	Com IRS *service* slc *legal, economic databases*
LUNAR	1972	Bolt Beranek Newman Cambridge MA	DEC PDP10 TENEX	Inst IRS nlq sqf ixf Woods[73]
MADAM	1970	MIT MacAIMS Proj. Cambridge MA	H6000 (MULTICS)	Dev DBMS *first* rel hlp (PL/1) stq rpg vrf Strnad[71]
MAGNUM	1975	Tymshare Cupertino CA	DEC PDP10	Com DBMS slc rel stq sch
MARKIV	1967	Informatics Canoga Park CA	IBM360/370 Univac900	Com FMS slc rpg tbq hie isf CODASYL[71A]
MARS	1969	Control Data Corp Sunnyvale CA	CDC6400	Com DBMS hie sch hlc(FORTRAN) sqf isf
MDBS	1980	Micro Data Base Syst Lafayette IN	Z80,8080-*based systems on* CP/M	Com DBMS slc stq net sch

Name	Year	Developer location	Computer	Type and features
MEDLARS *also* ELHILL	1963	National Lib Med Bethesda MD	IBM360	Inst SADBMS slc irq sqf isf bib Katter[75]. *also* ORBIT
MICRO-SEED	1980	Microsoft Bellevue WA	Z80,8080-*based systems*	ComDBMShlc(FORTRAN) net sch *based on* SEED
MODEL204	1972	Comp Corp of Am Cambridge MA	IBM360/370	Com DBMS hlc stq sch plf ixf(IFAM)
MORIS	1972	Polytechnico Milano Italy		Pro DBMS rel cal sch stq(COLARD) Bracchi in Klimbie[75]
MUMPS	1966	Mass Gen Hospital Boston MA	DEC PDP15,11 8080 ao	Com FMS slc hie trf Greenes[69]
MRDS *based on* RDMS	1978	Honeywell Inf Sys Minneapolis MN	H6000 L68 (MULTICS)	Com DBMS hlp(PL/1) rel stq(LINUS) rpg vrf
NOMAD	1975	National CSS Norwalk CN	IBM370-CMS	Com DBMS slc,hlc hie stq sch rpg
NYTIS	1970	New York Times New York NY	IBM360	Inst IRS slc txf ixf Baker[72]
OASIS	1971	Stanford Adm DP Stanford CA	IBM360/370	Inst SATDBMS hlc(COBOL) hie irq isf
ORACLE	1979	Relational Software Inc., Menlo Park CA	DEC VAX, IBM VM MVS DOS	DBMS hlc(COBOL PL/1 C) rel stq ixf cpr pri vie
OSIRIS	1973	Survey Res.Ctr.,Univ. of Michigan, Ann Arbor	IBM360/370	Inst SATDBMS sch sqf,isf rpg Rattenbury[74]
PLUS/4	1979	Century Analysis Pacheco CA	NCR 101 etc	Com FMS hlc
POLYPHEME (SIRIUS)	1977	I. N. Polytechnique Grenoble France	CII & IBM	Exp DDBMS pri LeBihan[80]
RAMIS	1967	Mathematica Princeton NJ	IBM360/370	Com DBMS hlc stq rpg
RAP	1975	Univ of Toronto Toronto Canada		Exp DBCMP rel Ozkarahan[77] Hawthorn[82]
RAPPORT	1978	Brit.Min.Def. & LOGICA,New York NY	*Any* FORTRAN-*based system*	Com DBMS hlc stq rpg rec
RDF	1967	Rand Corp Santa Monica CA	IBM360	Exp IRS slc nlq rel Levien[67]
RDMS *based on* MADAM	1971	Mass Inst of Tech Cambridge MA	H6000 (MULTICS PL/1)	Inst DBMS hlp rel stq rpg vrf Steuert in Rustin[74]

Name	Year	Developer location	Computer	Type and features
REGIS *early name:* RDMS	1972	General Motors Warren MI	IBM360-67	Dev DBMS hlp(PL/1) rel stq Joyce[76]
REL	1969	Calif Inst of Tech Pasadena CA	IBM360	Dev IRS niq rel *extendible* Thompson[69]
RELGRAF	1982	Adv.Rel.Techn.Inc. Menlo Park CA	PRIME OS *or* MPX	Com IRS stq rel cal vie txt gra sch ixf
RETRIEVE	1970	Tymshare Cupertino CA	XDS940	Com DBMS service slc stq 1-sqf; FMS(IML) *for* >2
RFMS	1971	Univ of Texas Austin TX	CDC6400	Dev DBMS hie stq sch Hardgrave[80]
RISS	1974	Forest Hosp.& MIT Des Plaines IL	DEC 11	DBMS 1-rel sch rpg sqf McLeod[75]
ROBOT	1973	Software Sciences Farnborough UK	ICL 1906 Univac9400	Com DBMS slc rpg tnf Palmer[75]
RS-1, *based on* PROPHET	1980	Bolt Beranek Newman Cambridge MA	DEC 11, VAX	SATDBMS slc *with* PL/1 tbq rpg grf sch
SAM *basis for* RM, XRM	1968	IBM Scientific Center Cambridge MA	IBM360*modified* 370	ExpFMSrelvrfSymonds[68] *later* RSS(SYSTEM R)
SAS	1972	Univ. N. Carolina & SAS Inst., Raleigh NC	IBM360/370	SATDBMS(*statistics*) slc stq rpg sqf
SBA *uses* QBE	1975	IBM Research Yorktown Heights NY		Dev DBMS rel tbq(*by example*) Zloof[77]
SCORE	1969	Programming Meth-ods, New York NY	Any COBOL system	Com FMS hlp(COBOL) hie tbq sqf isf
SDD-1	1978	Comp Corp of Am Cambridge, MA	DEC 10/20	Exp DDBMS slc rel sch DAPLEX Bernstein[81]
SEED	1978	Internat'l Data Base Syst., Philadelphia PA	DEC 11,10/20 IBM370 CDC6000	Com DBMS hlp(COBOL) hic(FORTRAN) sch rpg net(1973) rnf
SEQUITUR	1981	Pacific Software Berkeley CA	DEC-11,VAX Z-8000 systems	Com DBMS slc,hlc(C) rel tbq ixf rpg txt
SESAM	1973	Siemens München FRG	S4004	Com FMS stq rpg sqf isf
SHOEBOX	1970	MITRE Corp Bedford MA	IBM360	Exp SATDBMS slc txt plf stq Glantz[70]
SIBAS	1974	Shipping Res Svc Oslo Norway& Houston TX	IBM360/370 Univac1100 DEC PDP10	Com DBMS net hlp(COBOL) hlc sch Palmer[75]

Name	Year	Developer location	Computer	Type and features
SIR	1977	Scientific Information Retrieval, Evanston IL	IBM360/370 CDC 6000,Cyber	Com IRS stq stat.interface sch net pri
SOCRATE	1970	Univ of Grenoble, CII Louveciennes France	IBM360-370 CII Iris45,80	Dev, Com DBMS slc sch net stq vrf
SOLID	1967	Penn State Univ University Park PA	IBM360-40 -67	Exp DBMS slc cpr rec DeMaine[71]
SOURCE	1979	Telecomputing Co McLean VA	*multiple*	Com IRS *service* slc *business, consumer info, ads*
SPIRES	1967	Stanford Univ Stanford CA	IBM360-67, 370-168	Inst IRS, DBMS slc hie irq bib trf ixf pri Schroeder in Kerr[75]
SQL/DS	1981	IBM San Jose CA	IBM370 (DOS)	DBMS hlp(PL/1) rel stq ixf(VSAM) pri vie
STAIRS	1972	IBM Stuttgart FRG	IBM360/370 *with* CICS	Com IRS slc stq txt sqf drf isf pri
SWALLOW	1980	Mass. Inst. Tech. Cambridge MA		Exp DFMS obj
SYSIF	1970	CAP-SOGETI Paris 15 France	IBM360/370, CII Iris, Univac1100	Com IRS slc nlq sqf isf
SYSTEM 1022	1978	Software House Cambridge MA	DEC 10/20	Com DBMS slc,hlc ixf *timeshared services*
SYSTEM 2000 *or* S2000, S2K	1970	Intel-MRI Austin TX	IBM360/370 Univac1100 CDC6000	Com DBMS hlp(COBOL PL/1) hie stq isf Kroenke[78]
SYSTEM C	1981	Software Clearing House Cleveland OH	NCR Criterion	Com DBMS hlc net(1978) sch stq rnf rec pri
SYSTEM-R	1975	IBM Research San Jose CA	IBM370	ExpDBMShlp(PL/1)relstq (SEQUEL) vie Astrahan[76]
SYSTEM-R*	1981	IBM Research San Jose CA	IBM370 VMS	Exp DDBMS hlp rel sch stq vie
TDMS	1966	System Dev Corp Santa Monica CA	IBM360-50 (Adept)	Dev DBMS slc hie irq sch Bleier[68], CODASYL[71A]
TOD	1973	Stanford Univ. Stanford CA & ITTRI Chicago 16 IL	IBM360-50 67, 370 DEC VAX	Inst SATDBMS hlc(PL/1) sch irq gra ixf tnf cpr cip Wiederhold[75]
TOOL-IR	1974	Univ of Tokyo Tokyo Japan	HITAC8800	Inst IRS slc stq bib pri Yamamoto[75]

Name	Year	Developer location	Computer	Type and features
TOTAL	1971	Cincom Inc Cincinnati OH	IBM360/370 Univac 90,V70 CDC Cyber	Com DBMS hlc sch net (2 level:drf,rnf) DDICT Cagan[73]
		also on Siemens4004 Honeywell200 NCR Century		
TRAMP	1967	Univ of Michigan Ann Arbor MI	IBM360-67	Exp DBMS nlq rel Ash[68]
UCC TEN	1976	University Comp. Dallas TX	IBM360/370 IMS	Com DDICT slc rpg sch IMS
UNIDATA	1970	United Computing Kansas City MO	CDC6400	Com DBMS hie sch rpg gra
VISIFILE	1982	Visicorp San Jose CA	IBM PC, Apple	Com FMS slc ixf stq
WOODSTOCK	1979	Xerox Research Lab Palo Alto CA	Xerox Altos	Dev DFMS Swinehart[79]
ZETA *also* TORUS	1974	Univ of Toronto Toronto Ontario	IBM360/370	Exp DBMS hlc(PL/1) rel stq vie drf Tsichritzis[77]

Legend for type and features of database systems

alg	relational algebra	isf	indexed-sequential file
bib	bibliographic data	ixf	indexed files
BOMP	bill-of-materials processor	net	network database organization. (year indicates CODASYL standard)
cal	relational calculus		
cip	ciphering	nlq	natural language query capabilty
Com	commercial	obj	object based
cpr	compression	plf	pile file organization
DBCMP	database computer	pri	privacy protection
DBL	database language	Pro	proposed
DBMS	database-management system	QUS	query and update system
DDBMS	distributed DBMS	rec	recovery support
DDICT	data dictionary system	rel	relational database organization
Dev	developmental	rnf	ring or chain file organization
DFMS	distributed FMS	rpg	report generator
DPG	database program generator	SADBMS	single-application DBMS
drf	direct or immediate file organization	SATDBMS	single-application type DBMS
Exp	experimental	sch	schema
FMS	file-management system	slc	self-contained system
FS	file system	sqf	sequential file organization
gra	graphic data support	stq	statement-oriented query processor
hie	hierarchical database organization	tbq	tabular query processor
hlc	host-language system accessed by CALL	tnf	transposed files
hlp	host-language system with preprocessor	trf	tree-structured files
Inst	institutional	txt	textual data
irq	interrogative query processor	vie	support for multiple user views
IRS	information-retrieval system	vrf	virtual file support

Sept. 1983

Indexed File Access for ADA

ARTHUR M. KELLER
Stanford University, Computer Science Dept., Stanford, CA 94305 USA

SUMMARY

A proposed standard, but optional, library package for ADA for random access to files is defined. Various features are considered for inclusion. The file access features of several programming languages are compared and a proposed ADA specification is described. The specification is compatible with an existing portable file access system, FLASH, that has been used with PASCAL and other languages.

KEY WORDS Indexed sequential Multiple indexes Files ISAM Concurrent update Ada

CR Categories. D.3.3, H.3.2, H.2.2, H.2.3, E.2, D.4.3, D.3.4, D.4.5

INTRODUCTION TO FILE ACCESS ISSUES

Access to files is an important, but often neglected, consideration in programming language design. The specification for ALGOL 60 ignored files completely. In PASCAL, only sequential files are supported. Some other languages (*e.g.*, PL/I) provide a plethora of mutually incompatible file formats for different kinds of file access. The file access capabilities of ALGOL 68 allow access by location, as well as sequential access. [Tanenbaum 76]

Programming language designers often avoided file access because the issues were not well understood by them. Features provided usually require extensive support by runtime routines. For example, since programs normally deal with data in internal format (*e.g.*, binary), runtime routines usually convert between external format (*e.g.*, characters) and internal format. In addition, the underlying operating systems provide various levels of file system capabilities. If a programming language is to be portable, it must encompass this variety or ignore it. When programming language designers err by including too little functionality, users are forced to reinvent the wheel, often in mutually incompatible ways.

It is clear to us that a new higher standard level of capability should be included in new programming languages. First, file access issues are now well understood, as are the needs of the user community. Second, a basic level of file access primitives exists on all systems, and portable runtime routines can be written in higher level languages that support desired high level capabilities assuming only these low level primitives.

File access capabilities can be considered on a scale from sequential to random access. One extreme is to consider a file to be a *stream* of information. Stream files may only be read or written sequentially. They may have some structure. For example, *text* files are stream files that are subdivided into pages, lines, and characters (*e.g.*, by control characters), and they are suitable for printing. There is a file *pointer* that keeps track of where in the file the program is reading from. It is only possible to write at the end of the file.

Next on the scale is access by location. An address is associated with components of the file. There are two paradigms for access by location. The first is an extension of sequential

This work was supported in part by the Institute for Defense Analyses under DARPA project A-58, "Ada Language Studies." Dr. Victor Schneider, Principal Investigator, by the Office of Naval Research under Order Number N00014-78-F-0023, Contract No. LLL P0908303 for the S-1 project, Prof. Gio Wiederhold, Principal Investigator, by DARPA under Contract MDA903-77-C-0322 for the KBMS project, Prof. Gio Wiederhold, Principal Investigator, and by a National Science Foundation Graduate Fellowship.

access. Again there is a file pointer that remembers the address of the component of the file last read. The next read request will increment this pointer to determine the next component. The file pointer may be *repositioned* to another file address to cause subsequent access to start from the new location. The stream model assumes that sequential access is more frequent than repositioning. The second access paradigm assumes the opposite. Each read request specifies the file address of the component to be read. Here the unit of retrieval is important, and it is usually called a *record*. Access is simplified if all records are fixed length, since then addresses can be computed.

Both paradigms for access by location are functionally similar for read access, but lead to different programming language structures for their specification. In the first paradigm, writing can only occur at the end of the file, while in the second, any component may be rewritten by a new component of the same length. With the access by location model, files are often created sequentially. Insertions and deletions are not possible with this type of access, since they require extensive relocation of records, which affects the address information for record retrieval.

Access by value is third on the scale. Records are the unit of access, and they consist of *fields* that contain values. Usually, a unique value is associated with each record, and this value is called the *key*. For example, in a personnel file, if the employee name is the key, the information on a particular employee can be retrieved by specifying his or her name. The structure that supplies the address of a record given its key is called an *index*. Often the records are stored in ascending order by key, and this allows them to be read sequentially. Such a file is often called an indexed sequential file or an ISAM [IBM 1] file. Rewrites, insertions and deletions are supported. Since records may be moved as a result of an insert or deletion, access by address is usually not supported; access by key retrieves the correct record reliably.

The concept of a record is important to indexed access. Records for a given file often contain the same kind of information. In the personnel file example, the name, address, department, and salary for each employee could be stored in a record. An *attribute* is a field in each record that is used for a consistent purpose. The personnel file contains the name, address, department, and salary attributes. It is useful to allow records to be accessed using the value of any one of several attributes. For example, records in the personnel file could be accessed by name or department. Secondary indexes are used to access records through these alternate attributes. Duplication of values is allowed for secondary indexes; the primary index must have a unique value for each record.

More complicated structures for file access also exist. These usually fall into the category of support for databases, however. For example, it is often useful for records to cross reference each other or to records in other files. Hierarchical and network databases can be implemented that way. Such files can become arbitrarily complex, and are currently considered beyond the range of the programming language designer and implementer. But they are also beyond the range of a programmer without system support. Providing a consistent and portable cross referencing capability within the language would provide a tool to permit programmers to construct such database systems reliability within ADA.

Some have advocated the inclusion of databases in ADA [Hall 83]. There is significant disagreement about which database model to use, and how it should best be implemented. We feel that the file access support proposed here would make the task of building such a database system significantly easier, once the choice were made. Furthermore, there are many applications that need for the structured file support we propose but do not need a complete database system.

INTRODUCTION TO ISSUES FOR ADA

The question is where to draw the line for what to include in a programming language, such as ADA. Including too little reduces the usability of the language and requires users to reinvent the wheel each time. These wheels will have difference wheel and axle sizes, precluding portability to other vehicles. Including too much makes ADA virtually impossible to learn or implement. Language designers must be careful of *creeping featurism*. That means that features should only be included in a programming language if they are well understood and implementable. In addition, a feature should fill a definite need.

We feel *strongly* that some file access should be supported within ADA, and that the specifications for such file support should be at a level which is sufficiently well understood to be *implemented uniformly* on all ADA implementations that process stored data. Indexed sequential files should be supported by ADA implementations. Implementations for very small or imbedded computers may be able to do without such file access, but most other implementations will find such support exceedingly useful for applications programming and essential for data processing (*e.g.*, it is a universal feature in COBOL). It is important that each implementation that provides such support to do so in the same manner so that programs requiring indexed sequential files may be transported between these implementations. A useful compromise is to define a standard, but optional, package for accessing indexed sequential files in ADA.

The question then becomes what features should be included in this standard package. Simple indexed sequential files can be implemented using access by relative address, but the user should not have to interact with the system at such a low level. The standard for ISAM files is access by key, and the keys are typically of variable length. A drawback is the potential complexity of the required run-time support. Such runtime support exists, however, on many systems and is well understood [Wiederhold 83, Gray 78]. Furthermore, a portable file access system exists that provides such support and is written primarily in PASCAL. The motivation for inclusion is that there are many classes of applications that require random access to files. Database-oriented programs without random access are inconceivable. Data processing applications usually require something at least as powerful as ISAM. Embedded computer applications often need to manipulate large amounts of data. The question then becomes whether indexed sequential files are sufficient to support most databases and application programs.

COMPARISON OF FILE ACCESS CAPABILITIES OF SEVERAL LANGUAGES

This section is a consideration of the language features of several programming languages. PASCAL, ALGOL 68, and PL/I will be considered as examples of programming languages that are widely available and provide some file capabilities.

PASCAL

Standard PASCAL [Jensen 74] provides only sequential access to files. All files are treated as stream files. In addition, text files are supported by functions which automatically convert between internal and external format. There is also a controversy about how to do interactive input/output in this framework [Clark 79, Bron 79].

Some implementations of PASCAL also support limited random access to files. For example, CDC PASCAL [Jensen 74] supports a segmented file, a version of access by location. Files are divided into *segments*, all of the same size, implemented by disk pages. Operations include repositioning at the start of an arbitrary segment, completing the writing of a segment, and testing whether the file pointer is at the end of a segment. These features are not uniformly available and their use precludes portability to other implementations. For example, the DEC PASCAL from Hamburg does not support these functions [Kisicki 76].

ALGOL 68

ALGOL 68 [Tanenbaum 76] considers a file to be subdivided into pages, lines, and characters. Normally, access to files is done sequentially using a pointer. However, it is possible to reposition to any page, line, and character within the file. The effect of writing after repositioning the pointer to the middle of the file is not defined. This is because when writing, most tape drives destroy previously written information that follows the new location of the pointer, while disk drives typically retain it.

PL/I

PL/I [IBM 2] supports several types of files. There is a distinction between *stream* and *record* files. Stream files are like PASCAL text files. Record files are declared as using *sequential* access or *direct* (*i.e.*, random) access.

There are several options that a user of a record-oriented, direct file may specify. It may be *keyed* (*i.e.*, *indexed*); that is, records may be accessed by their key. The file may have one of six file organizations: consecutive, indexed, regional(1 3), or VSAM. *Consecutive* files may only be updated sequentially. *Indexed* files may be accessed randomly by key and are implemented using ISAM. There are three types of regional files. A *regional(1)* file contains fixed format records accessed by relative record number; there is no record key. A *regional(2)* file contains fixed format records with a key that are searched sequentially starting at a specified relative record number. A *regional(3)* file is the same as a *regional(2)* file except that variable length records are supported and a relative disk track number is used instead of a relative record number. Also supported are VSAM files [IBM 3], which are incompatible with any of the previous access methods. There are three categories of VSAM files and they permit the consecutive, indexed, and regional(1) types of access, respectively. Unfortunately, programs expecting a VSAM file cannot accept a similar, non-VSAM file instead. In practice, a file written using any of the 6 IBM access methods cannot be read by any other of the six.

Conceptually, such diversity is not necessary and some other implementations of PL/I (DEC VAX and PL/ACME) have provided common formats. The programmer using files with PL/I programs must be cognizant of many restrictions. Files created through one access method in some implementations are structured such that they may not be accessed using another access method. A program, however, can use several files, each using a different access method. Indexed VSAM files may have secondary indexes; that is, records can be retrieved based on one of several values contained in them. For example, information about a ship in a ship location file may be accessed by specifying either its name or its location.

Some access methods are more efficient in certain operations than others. Regional files are not supported by high level languages other than PL/I. For example, COBOL supports consecutive, indexed, and VSAM files, but not regional files [IBM 4]. Regional files are used, however, by some proprietary database management packages using assembler language routines that interface with IBM OS service routines.

REQUIREMENTS OF FILE ACCESS SUPPORT

Where should ADA's file access capabilities lie on the scale from sequential to random access?

Sequential access is clearly the minimum that should be supported. Access by location is easy to implement and is required by many applications [Knuth 72]. However, data processing applications and databases require access by key. Should each programmer have to implement a system for access by key? Or, rather, should a standard for access by key be defined and implemented for all programmers to use?

While not all implementations need to include access by key, it is important that those that do use the same package as an interface. The intent here is to define such a standard, but optional, package to be included in most ADA implementations.

Requirements for random access to files for ADA are derived from usage in application programs and in database systems. Stream files are not considered, they are assumed to exist in ADA. Functions required for databases are a superset of those needed for application programs, so only databases will be considered in this section. Consideration of application programs will be deferred until the proposed language features are specified.

Creating a standard for these features has several benefits. First, investment in a single, portable implementation will tremendously reduce costs by making it unnecessary to build such a system for new machines. Second, settling on a standard will specify what features exist on the minimal implementation, allowing for portable programs using these features. The access capabilities provided by the underlying operating system vary from system to system, so it is best to only assume the most basic functions provided on all systems. A portable file access system has been written that uses only these commonly-available basic functions (discussed further in section 5), and it can be easily adapted for use with ADA [Allchin 80]. We now enumerate basic requirements for the proposed package.

1. Access by key should be supported. A *key* is a value of any ADA type to be associated with the record to be retrieved. Use of a symbolic key provides independence of storage *mechanisms. Natural record structures often have components of variable length or* number, so variable length records should be supported. This would permit variant records to be used in indexed files. The user should access records symbolically (*i.e.*, by key value). It is desirable to be able to access records based on values of one of several attributes, so that multiple access paths may be needed for each record. Access by key instead of access by location permits multiple access paths and variable length records.
2. The unit of retrieval should be a collection of related fields, called a *record*. Concurrency and update considerations preclude use of the repositioning paradigm (access by repositioning the file pointer in a stream file) with its attendant sequential access. *Concurrent access* requires that data be locked from improper simultaneous access. This is not possible in stream files, as it is not known in advance how much of the file will be read when the file pointer is repositioned. The unit of access should also be the unit of concurrency control, and this is facilitated by the use of records for both. Concurrency is discussed further in number 4. Another important problem with the repositioning paradigm is the calculation of the starting location of a record when variable-length records are used. A problem with stream files is that programs must necessarily be allowed read and write variables of different types in the same file. When a file consists of records all of one type, access by record permits the type-checking facilities of ADA to reduce the incidence of error.
3. Multiple indexes should be supported. This allows a record to be accessed by more than one key. For example, in a ship location file, a record could be retrieved by specifying the name of the ship, its location, its country of registry, or other identifying information. Indexed sequential access methods that are available do not universally provide multiple indexes. For example, ISAM does not support multiple indexes. Multiple indexes were not in the original implementation of VSAM, but were added in later; a better design would

be to include multiple indexes in the original design. Once indexed sequential access is provided, the incremental programming cost of including secondary indexes is minor, while the increase in capability is great. To support secondary indexes, additional index and referencing structures are required. All indexes need to be updated when records are inserted, modified, or deleted. Also, a *file pointer* is needed that keeps track of the last record accessed and the index used for that access. This means that both the key and index must be specified to access a record, and the next record means the record with the next higher value for that index.

4 Concurrent and simultaneous access should be supported. Consistency of the file can be maintained in the face of concurrent updates by locking of records. This can be done by specifying the intention to modify on the read request. The record is then locked when it is read until it is updated or deleted, or the intention to modify is reneged. Of course, records may be read without declaring an intention to modify, in which case the record will be read when there are no conflicting locks outstanding [Garcia-Molina 82]. However, the record may not be updated then, as another user could be updating it. It is also reasonable to implement non-audit reads, which would not involve any locks; such access, however, could lead to an inconsistent state of the file. Because ADA supports *multitasking* (*i.e.*, concurrent execution of procedures), it is important to allow multiple tasks to access the same file simultaneously.

Some implementations may permit multiple records to be locked by the same access path simultaneously. The number of such simultaneous requests is implementation dependent. This permits several records affected by a transaction to be read before any of them are updated. Furthermore, when records are updated, their locks may be held until the transaction is terminated (*e.g.*, at commit time [Gray 79, Gray 80]).

These requirements could easily be extended. However, the features described here are sufficient to build database systems (hierarchical, network, or relational) and other systems requiring sophisticated access to large amounts of data. For any set of requirements the implementation will become more complex. The existence of a file access system that is a reasonable compromise between compactness and power can suggest a compromise of requirements and implementation cost.

ADDITIONAL FUNCTIONS NEEDED

There are several functions that should be available for record-oriented files. First, functions are needed that establish and discontinue an association between an internal file variable and an operating system file. Retrieval support should include the capability of reading a record with a given key and reading the successor record. In addition, repositioning of the file pointer to an arbitrary key should be allowed. Capability to update or delete existing records should also exist. There should be functions whereby new records may be inserted in the middle of the file or appended to the end of the file.

SPECIFICATIONS

The specifications of the individual procedures follow the overall specification.

```
generic
    type INDEXED_FILE is limited private;
    type RECORD_TYPE is private;         -- type of record in file
    type FIELD_DESCRIPTORS is
        array (INTEGER range <>) of FIELD_DESCRIPTOR;
    type TUPLE_LIST_TYPE is
        array (INTEGER range <>) of TUPLE_ID_TYPE;
    MAX_KEY_LENGTH: constant INTEGER := 125;     -- implementation defined
    MAX_FIELDS: constant INTEGER := 128;         -- implementation defined
    MAX_RECORD_SIZE. constant INTEGER := 2550;  -- implementation defined



package INDEXED_IO is

    -- types of frequently used parameters
    type MODIFY_INTENTION_TYPE is
        (WILL_MODIFY,           -- tuple is locked on read until
                                -- updated or released
                                -- no one else is allowed access
        READ_LOCK,              -- allow others to read but not
                                -- to update
        WILL_NOT_MODIFY);       -- otherwise



    type ACCESS_TYPE is
        (DO_INPUT,              -- read-only access
        DO_INSERT,              -- inserts only, reads not allowed
        DO_UPDATE,              -- all kinds of access allowed
        DO_CREATE);             -- create file, then DO_UPDATE



    type TUPLE_ID_TYPE is private;
                                -- tuple ID for quick access



    type EXCLUSIVE_TYPE is
        (EXCLUSIVE,             -- no one else may access file
        NORMAL,                 -- any number of readers or 1 writer
        ONE_WRITER,             -- readers, 1 writer, or both
        SHARED),                -- any number of readers and writers



    -- types used to specify fields on OPEN
    type FIELD_TYPE is
        (FIXED_LENGTH,          -- field always same length
        VARIABLE_DELIMITED,     -- field always ends with same character
        VARIABLE_LENGTH);       -- field preceded by a count byte
```

```
type FIELD_DESCRIPTOR (FIELD_CLASS: FIELD_TYPE) is
    record
        INDEX_NUMBER: INTEGER;
                                  -- used to specify which index to search
        CLUSTERED_INDEX: BOOLEAN;
                                  -- specifies physical ordering of tuples
        UNIQUE_KEYS: BOOLEAN;
                                  -- keys must be unique in file
        case FIELD_CLASS is
            when FIXED_LENGTH =>
                FIELD_LENGTH: INTEGER range 1..MAX_RECORD_SIZE;
            when VARIABLE_DELIMITED =>
                FIELD_DELIMITER: CHARACTER;
            when VARIABLE_LENGTH =>
                null;
        end case;
    end record;

-- visible procedures for INDEXED_IO
procedure OPEN      -- opens a file for future access
    (FILE_ID         : out INDEXED_FILE;
    FILE_NAME        : in STRING;
    OPEN_ACCESS      : in ACCESS_TYPE := DO_UPDATE;
    OPEN_EXCLUSIVITY: in EXCLUSIVE_TYPE := NORMAL;
    OPEN_FIELDS      : in FIELD_DESCRIPTORS);

procedure CLOSE     -- closes a file releasing access
    (FILE_ID         : in INDEXED_FILE;
    DELETE           : BOOLEAN := FALSE);

procedure READ_KEY  -- reads a record using a key
    (FILE_ID         : in INDEXED_FILE;
    INDEX_NUMBER     : in INTEGER range 1..MAX_FIELDS;
    KEY_EXPRESSION   : in STRING;
    RECORD_VARIABLE  : out RECORD_TYPE;
    MODIFY_INTENTION: in MODIFY_INTENTION_TYPE := WILL_NOT_MODIFY;
    TUPLE_ID         : out TUPLE_ID_TYPE);

procedure READ_NEXT -- reads the next record for a given index
    (FILE_ID         : in INDEXED_FILE;
    INDEX_NUMBER     : out INTEGER range 1..MAX_FIELDS;
    KEY_EXPRESSION   : out STRING;
    RECORD_VARIABLE  : out RECORD_TYPE;
    MODIFY_INTENTION: in MODIFY_INTENTION_TYPE := WILL_NOT_MODIFY;
    TUPLE_ID         : out TUPLE_ID_TYPE);

procedure READ_TUPLE -- reads the record given its tuple ID
    (FILE_ID         : in INDEXED_FILE;
    RECORD_VARIABLE  : out RECORD_TYPE;
    MODIFY_INTENTION: in MODIFY_INTENTION_TYPE := WILL_NOT_MODIFY;
    TUPLE_ID         : in TUPLE_ID_TYPE);
```

```
procedure UPDATE    -- rewrites a previously read record
    (FILE_ID        : in out INDEXED_FILE;
    RECORD_VARIABLE : out RECORD_TYPE;
    TUPLE_ID        : out TUPLE_ID_TYPE;
    RELEASE_LOCKS   : in BOOLEAN := TRUE);


procedure DELETE    -- deletes a previously read record
    (FILE_ID        : in out INDEXED_FILE;
    TUPLE_ID        : out TUPLE_ID_TYPE;
    RELEASE_LOCKS   : in BOOLEAN := TRUE);


procedure INSERT    -- inserts a new record
    (FILE_ID        : in out INDEXED_FILE;
    RECORD_VARIABLE : in RECORD_TYPE;
    RELEASE_LOCKS   : in BOOLEAN := TRUE);


procedure POINT     -- positions a file pointer for READ_NEXT
    (FILE_ID        : in INDEXED_FILE;
    INDEX_NUMBER    : in INTEGER range 1..MAX_FIELDS;
    KEY_EXPRESSION  : in STRING);


procedure GET_TUPLE_IDS -- gets list of tuple IDs for READ_TUPLE
    (FILE_ID        : in INDEXED_FILE;
    INDEX_NUMBER    : in INTEGER range 1..MAX_FIELDS;
    KEY_EXPRESSION  : in STRING;
    TUPLE_LIST      : out TUPLE_LIST_TYPE;
    NUMBER_RETURNED : out INTEGER;
    START_FROM      : in INTEGER := 1);


procedure RENEGE    -- releases locks on record
    (FILE_ID        : in INDEXED_FILE;
    TUPLE_ID        : in TUPLE_ID_TYPE);


procedure END_TRANSACTION
                    -- forces all records to stable storage and
                    -- releases all locks on records
    (FILE_ID        : in out INDEXED_FILE);


procedure ERROR_MESSAGE
                    -- returns a descriptive error message
    (FILE_ID        : in INDEXED_FILE;
    MESSAGE         : out STRING);


-- exception handlers
procedure ERROR_MESSAGE     -- returns description of last error
    (FILE_ID        : in INDEXED_FILE;
    ERROR_MSG       : out STRING);
    -- list of exceptions
    END_OF_FILE,          -- last read beyond data
    NOT_OPEN,             -- attempt to do anything but open a unopened file
    USE_ERROR,            -- attempt to perform an unpermitted operation
                          -- eg, write into a file opened for input
                          -- update without reading first
    RECORD_NOT_FOUND,     -- read record with specific key failed
    DUPLICATE_RECORD,     -- duplicate record on some index with unique keys
    UNAUTHORIZED_ACCESS,-- access request not as authorized by open
    INTERNAL_ERROR,       -- error detected in implementation
    OTHER_ERROR:          -- unclassified error
        exception;

private
    -- declarations of private types and data structures

end INDEXED_IO;
```

DESCRIPTIONS OF THE INDIVIDUAL PROCEDURES

This section and its subsections describes in detail what the individual procedures of the INDEXED_IO package do. An overview of the implementation appears later.

The INDEXED_IO package needs to know what fields have indexes and what their formats are. For an old file, this information is maintained internally with the file. However, for a new file, this information must be supplied by the creator of the file when it is first opened. Therefore, the array of type FIELD_DESCRIPTORS describes these fields to the package. It is unfortunate that the package cannot use the definition of the record that is being stored in the file, but the definition is available to the compiler at compile time but the package uses the information at run-time. A similar problem is faced by an interactive debugger; a feature that would solve both problems is the ability to inquire about the structure of records and types of variables.

The array of type TUPLE_LIST_TYPE is used only in the procedure GET_TUPLE_IDS and is described there.

Concurrency control requires that records be locked for the stretch of time between reading and rewriting [Eswaran 76, Gray 76]. Therefore, when a record is read, the intention to rewrite it must be declared. Furthermore, it may be important to be immune from further changes to the record, and so a request to hold a read-only lock for the record may be issued. The type MODIFY_INTENTION_TYPE allows the necessary distinctions.

The type ACCESS_TYPE is used only during OPEN and is described there.

Frequent and repeated access to the same record in the file may be improved by providing a quick method for accessing it. In databases, such a quick method for accessing involves the use of a *tuple ID* to refer to the tuple desired. The tuple ID is independent of the location of the record, so it remains valid even this record is changed, or other records are inserted or deleted. It is vitally important that the tuple ID never be used to obtain a different record than the one originally associated with it. This precludes tuple IDs for any file from being reassigned when a record is deleted. A file which has undergone many insertions and deletions will have many tuple IDs which are being maintained as not referring to any record, but it will also likely be unbalanced. Reorganization of the file will rebalance it, as well as reassigning new tuple IDs to every record.

The type **FIELD_DESCRIPTOR** is used only in the type **FIELD_DESCRIPTORS** as described above.

PROCEDURE OPEN

The procedure **OPEN** is used for establishing an access path to the file. The **FILE_NAME** is a string containing the name of the file. This name is operating system-dependent, and has the semantics of **NAME** in the package **INPUT_OUTPUT** as described in chapter 14 of the ADA reference manual [DoD 80]. In particular, some operating systems may support inclusion of device, directory, and protection specifications as part of the file name specification. The parameter **OPEN_ACCESS** describes the kind of access that the user desires to perform against the file. Access to the file not in accordance with the access specified at open time will result in an error exception, **UNAUTHORIZED_ACCESS**.

File sharing is an important concept in database access. Different transactions often want to access the same files simultaneously, but in a controlled manner. Requiring that all such access be done by one ADA task would create an incredible bottleneck drastically reducing the efficiency of the system. On the other hand, many efficiencies can be introduced by taking advantage of the knowledge of the type of sharing to be used. As a result, the permitted level of sharing should be specified with the open request.

The permitted level of sharing is described in **OPEN_EXCLUSIVITY**. The following table details sharing permitted by different users. "Y" means the access combination is allowed; "N" means it isn't.

	THEN	reader				writer			
US		Exclusive	Normal	One_Writer	Shared	Exclusive	Normal	One_Writer	Shared
reader	Exclusive	N	N	N	N	N	N	N	N
	Normal	N	Y	Y	Y	N	N	N	N
	One_Writer	N	Y	Y	Y	N	N	Y	Y
	Shared	N	Y	Y	Y	N	N	Y	Y
writer	Exclusive	N	N	N	N	N	N	N	N
	Normal	N	N	N	N	N	N	N	N
	One_Writer	N	N	Y	Y	N	N	N	N
	Shared	N	N	Y	Y	N	N	N	Y

When the file is created, the additional parameter **OPEN_FIELDS** is used to describe the layout of the file. A file is considered to consist of records, each of which consists of several fields. The fields are contiguous and are of various types. Any field that appears in all records may have an index associated with it. Such an index permits searching for records with particular values for that field. One index should be designated as a clustered index indicating that records are to physically stored in order according to that index. This improves the performance of sequential access to the file using that index. Choosing a clustered index has no semantic effect, but it does improve performance. In addition, any index may be specified as having unique keys. When records are inserted or updated, all indexes with unique keys are checked, and if any duplicates are found the request is aborted.

it would be preferable if the programmer did not have to specify the data type twice: when the data type is declared and to the file access package. This would require the feature that generic packages could determine the structure of a structured variable passed to it. Currently, the only thing that generic packages can do with the unspecified type is use operators common to all types supported. In particular, that does not allow a generic package to vary its behavior based on the implementation of the type passed to it. Such a feature would also permit interactive debuggers to access the components of structured types upon user request. This concept is explored further elsewhere [*Keller 83b*].

PROCEDURE CLOSE

The procedure **CLOSE** terminates an access path to a file. The **FILE_ID** is no longer valid to access a file until it is used in an **OPEN** request. Other tasks or users having the file open may continue to access the file; also any other **FILE_ID**s referring to the same file may continue to be used.

The file will be deleted if **DELETE** is true and the file is not opened by any other user, task, or **FILE_ID**.

PROCEDURE READ_KEY

The procedure **READ_KEY** is used to read a record from the file having a particular key (**KEY_EXPRESSION**) for the specified index (**INDEX_NUMBER**). If the record is to be modified or protected from modification by others, **MODIFY_INTENTION** should be set to the appropriate value. (See section 4.1.6 (**UPDATE**) for a description of locking.) Subsequent access to the same record can be improved by using the returned **TUPLE_ID**. The file cursor is set so that the **READ_NEXT** procedure will read the next record for that particular index. (See the next section for a description of the file cursor.) Note that to read every record with a particular key (for a non-unique index), **READ_KEY** should be used to read the first record and **READ_NEXT** to read the others until the key changes; unless the file changes, **READ_KEY** will return the same record given the same key and index. Unsuccessful **READ_KEY** requests will not update the file cursor.

PROCEDURE READ_NEXT

The procedure **READ_KEY** is used to read a the next record from the file for a particular index number. The key and index number are returned in **KEY_EXPRESSION** and **INDEX_NUMBER**, respectively. The parameters **MODIFY_INTENTION** and **TUPLE_ID** are as in **READ_KEY**.

There is an file cursor associated with each open **FILE_ID**. This cursor is set to the key and index used in **READ_KEY** and **POINT** requests. The cursor is automatically incremented by the **READ_NEXT** procedure to read records that follow according to that index. To read every record with a particular key (for a non-unique index), **READ_KEY** should be used to read the first record and **READ_NEXT** to read the others until the key changes. To read every record starting with a key prefix, use the **POINT** procedure to specify the prefix and index number, and then **READ_NEXT** to read records until the key returned no longer has that prefix.

PROCEDURE READ_TUPLE

The **READ_TUPLE** procedure is used to read a record with a particular **TUPLE_ID**. Candidate values of **TUPLE_ID** may be obtained by using the **READ_KEY** or **READ_NEXT** procedure to obtain the tuple ID associated with a particular record. **GET_TUPLE_IDS** may be used to obtain a list of tuple IDs associated with all records with a particular key and index number. The parameter **MODIFY_INTENTION** is as in **READ_KEY**.

PROCEDURE UPDATE

The procedure UPDATE may be used to replace the record previously read. The TUPLE_ID is not changed as a result of the UPDATE request regardless of which indexed field have changed.

The record must have been read specifying MODIFY_INTENTION as WILL_MODIFY. That causes the record to be locked from reading or updating by other users, tasks, or FILE_IDs. If the record is not to be updated, RENEGE should be used to release the lock so that others may now access it. The parameter RELEASE_LOCKS should be set to true if the record is to be released for access by others. The record is remains locked from changes by other users, tasks, or FILE_IDs if RELEASE_LOCKS is false, in which case the locks should be released explicitly by another request, such as UPDATE or RENEGE. An END_TRANSACTION request will release all outstanding locks to records in the file.

PROCEDURE DELETE

The procedure DELETE may be used to delete a previously read record. The TUPLE_ID becomes invalid for accessing any record; it will not be reused to access another record in this file. See the previous section for a description of the use of RELEASE_LOCKS.

PROCEDURE INSERT

The procedure INSERT is used to insert a new record into the file. The TUPLE_ID is set for subsequent retrieval of the record. See section "PROCEDURE UPDATE" for a description of the use of RELEASE_LOCKS.

PROCEDURE POINT

The procedure POINT sets the file cursor for use in subsequent READ_NEXT requests. See section 4.1.4 (READ_NEXT) for a description of the file cursor. To set the file cursor so that all records of the file will be read, use a null key.

PROCEDURE GET_TUPLE_IDS

The procedure GET_TUPLE_IDS is used to obtain a list of tuple IDs associated with all records having a particular key and index number. Since this list may be rather large, START_FROM may be used to obtain the next set of tuple IDs by setting it to one more than the value of NUMBER_RETURNED by the previous request. Of course, this need only be done if the value of NUMBER_RETURNED was the number of elements of TUPLE_LIST.

This procedure may be used to retrieve records satisfying a list of constraints. For example, suppose we want to read all records that have key FOO for index 1 and key BAR for index 2. We first GET_TUPLE_IDS for FOO and BAR and then read using READ_TUPLE all records whose tuple IDs are in both lists. Note that the list returned is in arbitrary order, so the program should sort them before comparing the lists.

PROCEDURE RENEGE

The procedure RENEGE is used to release locks on a particular record. See section 4.1.6 (UPDATE) under the description of RELEASE_LOCKS for a description of locking.

PROCEDURE END_TRANSACTION

The procedure END_TRANSACTION is used to release all record locks after forcing all records to stable (e.g., disk) storage.

PROCEDURE ERROR_MESSAGE

The procedure **ERROR_MESSAGE** returns a descriptive, system-dependent error message after an exception occurs.

IMPLEMENTATION

Definition and implementation of keyed access, record-oriented file support for ADA is greatly simplified by the existence of a portable file access system, *FLASH* [Allchin 80, Keller 83a]. The FLASH file system supports all of the functions described in the previous two sections. It is written in PASCAL, enabling it to run on a variety of machines. Operating system-dependent code is isolated in several routines (currently implemented in DEC System-20 MACRO, VAX UNIX C, and IBM 370 assembler) that consist of about 250 lines of code; the remainder of FLASH consists of approximately 7000 lines of PASCAL code. FLASH can be used with PASCAL, FORTRAN, and INTERLISP, so no problems are anticipated using it with ADA.

The FLASH file system uses a B^+-tree [Bayer 72, Comer 79, Knuth 73] implementation. The unit of transfer is equal to a block, and, for efficiency on virtual storage systems, the block size is set equal to the page size. Except for a current limit that the record size must be less that the block size (spanned records have not yet been implemented), the user of FLASH is not aware of its parameters. The locking algorithm is designed to allow maximal concurrency [Bayer 77, Guibas 78] when supported by the operating system. Reliability is enhanced through the technique of leaf-first updating [Schwartz 73]. The buffer management strategy uses a modified least-recently-used (LRU) scheduling discipline. FLASH must know about the formats of the fields of the record for indexing purposes; as a result, a very flexible record format was defined and is supported.

Few assumptions are made of the capabilities of the underlying operating system. A directory of files is expected to exist, and the required functions are: directory management, open and close files, allocate a buffer, and read, write, allocate, release, and flush a file block. In addition, the ability to lock on symbolic names (e.g., ENQ and DEQ [BBN 73]) is needed to support shared access. As these functions exist on most systems, it is unnecessary to write them. For example, one reason for the large size of VSAM is that it attempted to reimplement most of these functions since the operating system does not provide them in a satisfactory form.

CONCLUSION

Random access files are an important feature to include in ADA, if the language is to support portable data processing applications. The proposal for random access files described provides the needed file system capability. It would facilitate the construction of database systems and application programs that handle large amounts of data. The feasibility of an efficient implementation is demonstrated by an existing portable file access system - FLASH. Steelman requires random access files, and this proposal satisfies these requirements.

CREDITS AND ACKNOWLEDGMENTS

Gio Wiederhold gave constructive criticism. Victor Schneider provided encouragement. John Hennessy commented on an early draft of this paper.

REFERENCES

[ADA 1] "Preliminary ADA Reference Manual," *Sigplan Notices* 14, 6 (June 1979), part A.

[ADA 2] "Rationale for the Design of the ADA Programming Language," *Sigplan Notices* **14**, 6 (June 1979), part B.

[Allchin 80] J. Allchin, A. M. Keller, and G. Wiederhold, "FLASH: A Language-Independent, Portable File Access System," in *Proc. Int. Conf. on Management of Data*, (Santa Monica), ACM SIGMOD, May 1980.

[Bayer 72] R. Bayer and C. McCreight, "Organization and Maintenance of large ordered indexes," *Acta Inf.* **1**, 3 (1972), 173–189.

[Bayer 77] R. Bayer and M. Schkolnick, "Concurrency of operations on B-trees," *Acta Inf.* **9**, 1 (1977), 1–21.

[BBN 73] Bolt Beranek and Newman, Inc., *Tenex jsys manual*, Cambridge, MA, September 1973.

[Bron 79] C. Bron and E. Dijkstra, "A Discipline for the Programming of Interactive I/O in PASCAL," in *Sigplan Notices* **14**, 12 (December 1979), 59–61.

[Clark 79] R. Clark, "Interactive Input in PASCAL," in *Sigplan Notices* **12**, 2 (February 1979), 9–13.

[Comer 79] D. Comer, "The Ubiquitous B-tree," in *Comput. Surv.* **11**, 2 (June 1979), 121–137.

[DoD 78] *Requirements for High Order Computer Programming Languages: "Steelman,"* Dept. of Defense, 1978, 8B.

[DoD 80] *Reference Manual for the ADA Programming Language, Proposed Standard Document*, United States Department of Defense, July 1980.

[Eswaran 76] K. Eswaran, J. Gray, R. Lorie, and I. Traiger, "On the Notions of Consistency and Predicate Locks in a Relational Database System," in *Comm. ACM* **19**, 11, November 1976, pp. 624–634.

[Garcia 82] H. Garcia-Molina and G. Wiederhold, "Read-Only Transactions in a Distributed Database," *ACM Trans. on Database Systems*, **7**·2, June 1982.

[Gray 76] J. Gray, R. Lorie, G. Potzolu, and I. Traiger, "Granularity of Locks and Degrees of Consistency in a Shared Data Base," in *Modelling in Data Base Managment Systems*, Nijssen (editor), North Holland, 1976, pp. 365–394.

[Gray 78] J. Gray, "Notes on Data Base Operating Systems," in *Operating Systems: An Advanced Course*, Bayer, Graham, and Seegmuller (editors), Springer-Verlag, 1978, pp. 393–481.

[Gray 79] J. Gray, *et al*, *The Recovery Manager of a Data Management System*, IBM Research Report RJ 2623, IBM Research Laboratory, San Jose, CA, August 1979.

[Gray 80] J. Gray, *A Transaction Model*, IBM Research Report RJ2895, IBM Research Laboratory, San Jose, CA, August 1980.

[Guibas 78] L. Guibas and R. Sedgewick, "A Dichromatic Framework for Balanced Trees," in *Proc. 19th Annual Symp. on Foundations of Computer Science*, Ann Arbor, Mich., 1978, 8–21.

[Hall 83] P. Hall, "Adding Database Management to Ada," in *ACM SIGMOD*

Record, **13**,3, April 1983.

[IBM 1] *OS/VS Data Management Services Guide*, Order No. GC26-3783, IBM, Armonk, NY.

[IBM 2] *OS PL/1 Checkout and Optimizing Compilers: Language Reference Manual*, Order No. GC33-0009, IBM, Armonk, NY.

[IBM 3] *OS/VS Virtual Storage Access Method (VSAM) planning guide*, Order No. GC26-3799, IBM, Armonk, NY.

[IBM 4] *OS/VS Cobol: Language Reference*, Order No. ****-****, IBM, Armonk, NY.

[Jensen 74] K. Jensen and N. Wirth, *Pascal user manual and report*, Springer-Verlag, New York, 1974.

[Keller 83a] A. M. Keller, "Implementation of a File With Multiple Indexes Using B-Trees," in preparation.

[Keller 83b] A. M. Keller, "Making the Symbol Table Useful for Programs at Runtime," in preparation.

[Kisicki 76] E. Kisicki and H. Nagel, *Pascal for the DEC System-20*, Institut fuer Informatik. Hamburg, Germany, 1976.

[Knuth 72] D. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[Knuth 73] D. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, second edition, 1973.

[Schneider 80] V. Schneider, "Proposed Ada I/O Revision," personal communication.

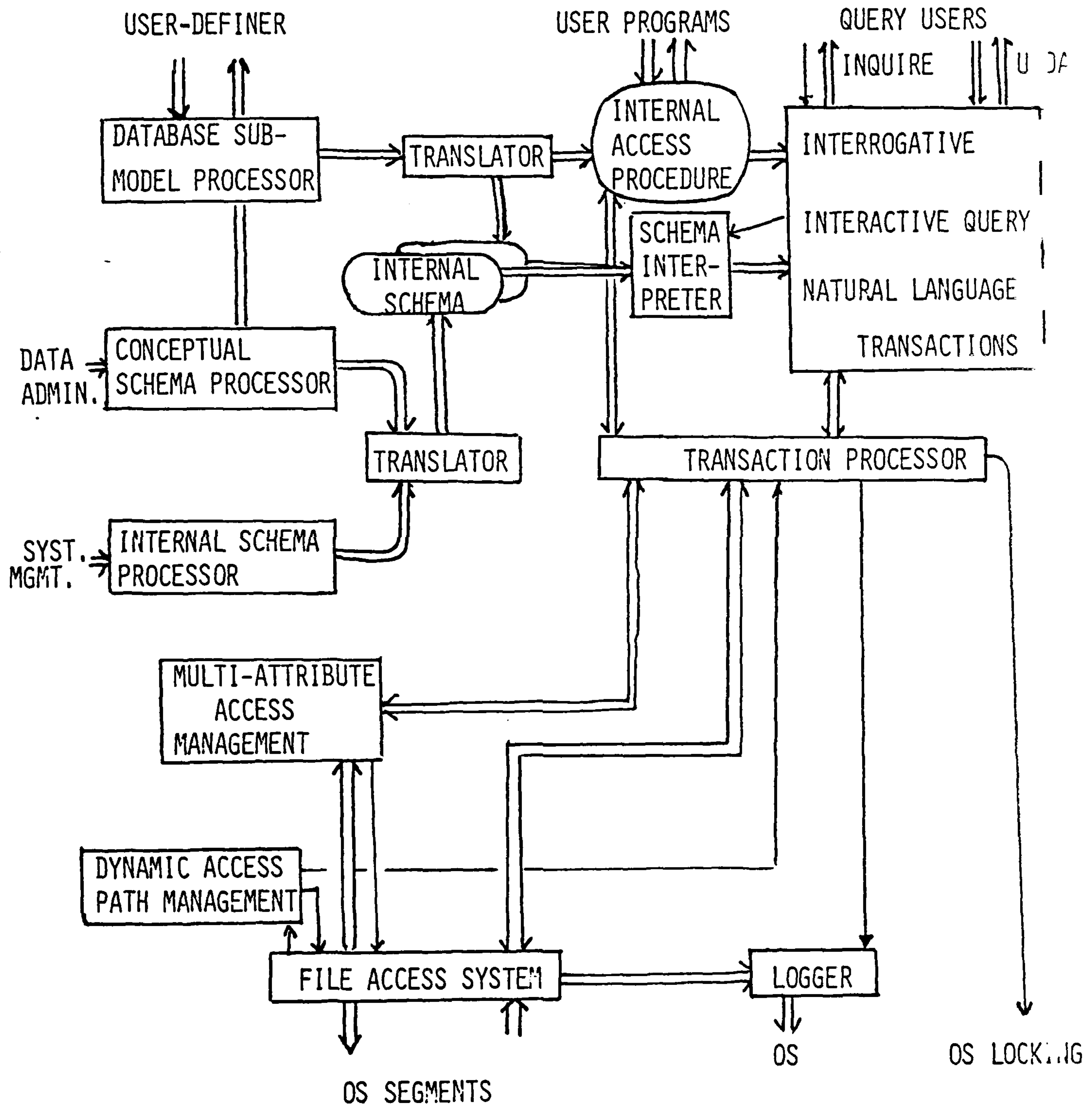
[Schwartz 73] M. Schwartz, *A storage hierarchical addressing space for a computer file system*, Ph. D. dissertation, Case Western Reserve University, Cleveland, Ohio, January 1973. Also Andrew R. Jennings Computation Center, Case Western Reserve University, Cleveland, Ohio, Jennings Report 1144.

[Tanenbaum 76] A. Tanenbaum, "A Tutorial on Algol 68," in *Comput. Surv.* **8**, 2 (June 1976), 155 190.

[Wegner 80] P. Wegner, *Programming with ADA: An Introduction by Means of Graduated Examples*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1980.

[Wiederhold 83] G. Wiederhold, *Database Design*, McGraw Hill, New York, second edition, 1983.

DIAGRAM OF MODULES



Database Technology Review and Development Estimates

Prepared for
The Institute for Defense Analysis

Prepared by
The Computer Corporation of America
September 14, 1983

1. Overview

Database management provides the technology for managing enormous amounts of diverse data required for large military operations. The primary objective of this technology is to provide efficient and reliable access to the shared planing, logistics, operations, and intelligence data required for world-wide command and control. The capabilities normally associated with database technology include:

1. Database definition - the capability to define and redefine the logical and physical data structures independent of any specific application program.

2. Database retrieval and update - the capability to retrieve and update from a shared database.

3. Authorization control - the capability to control which users can update and retrieve which data.

4. Multiple user updates - the capability to maintain the integrity of the database while it is being simultaneously updated by multiple users.

5. Back-up and recovery - the capability to save copies of the database, to save changes made to the database since the last copy, and to restore the database in case of system or media crash.

The challenge will be to provide these capabilities with enough flexibility to support dynamically changing mission and performance requirements and to intelligently utilize these capabilities to provide selective access to timely and critical information.

In order to provide the needed flexibility and performance, several levels of interfaces to the database management technology must be provided:

- Programming language interface - the data must be accessible via programming language (Ada) in order to directly and rapidly access the data for numerical analysis, simulations, graphics algorithms and other application programs.

- Query language interfaces - the data must be accessible via a query language to provide powerful and flexible access to interactive users for ad-hoc queries.

- Screen oriented interfaces - the data must be accessible through screen oriented form and menu selection interfaces to provide access to non-computer specialists and changing personnel.

- Bulk load and unload utilities - these utilities allow data to be loaded and unloaded between files so it can be rapidly used with existing programs.

- Report Writer Utilities - these utilities provide for automatic formatting of the data for repetitive reports.

- Application generators - these utilities provide for automatic generation of forms and database actions for frequent and repetitive database updates and retrievals.

In order to provide an overview of the state of the art of the database technology and produce a development cost estimate for that technology, we studied the capabilities, interfaces, and architectures of three commercial Database Management Systems and one prototype system being developed in Ada. The case studies were conducted for the following systems:

1. Model 204 - CCA's commercial system with relational capabilities that is available on IBM systems.

2. Oracle - a commercially available relational system that runs on 32 b[illegible] mini computers.

3. DBMS-20 - a commercially available DBTG network system.

4. LDM - a prototype system being implemented (95% complete) in Ada by CCA.

Based on these studies and our general database expertise, we identified the key performance factors and estimated development levels of effort.

The performances of different database management systems and technologies is difficult to compare due to the extreme dependence on the specific applications and computer systems on which the DBMS is run. The typical metrics used for DBMS performance measurements include:
- number of single record updates per second, and
- retrieval rates for large quantities of data.

Unfortunately those times are extremely dependent on factors such as the sizes of the databases, the number of indexes on the records being updated, the complexity of the database query and the class of machine on which the performance measurements are to be made. The retrieval rates are extremely difficult to compare since even for one system the rates can differ by over an order of magnitude depending on the number of files that had to be accessed to collect the required data.

The single record updates per second can be more directly compared since factors such as the number of access paths can be controlled. However, time did not permit the execution of controlled experiments for a number of different systems. Furthermore, the performance results that were obtained were for vastly different classes of hardware. The results obtained were consistent with the general range of update rates shown in Table 1.

Table 1

System Size	Update Rates
Personal Computers	.5 - 1 / second
Small mini computers	1 - 4 / second
Super mini computers	2 - 10 / second
Large mainframe	10 - 25 /second

It is important to understand the range of performance required for a given set of applications and then to select or design the hardware required. There are many unfortunate examples of the hardware selection preceding the applications definition. The result has been that the required database performance cannot be achieved.

The case studies were very useful, however, for verifying our estimates of the levels of effort required to use and develop database management for Ada. The levels of effort for different alternatives are summarized in Table 2.

Table 2

System	Prototype Level of Effort (years)	Production Level of Effort (years)	Elapsed Time (years)
Use Existing DBMS	1.5	-	.75
Integrated File System	4	8	1.5
Full DBMS	12	25	3.0

The first effort would produce an Ada interface to an existing relational system. This approach assumes the existence of a DBMS on the target system and interprocess communication capability (IPC) provided through the Ada runtime system.

The second effort would provide a library of integrated file access methods which could be incorporated into Ada programs. The integrated library is required in order to effectively support the authorization, multi-user, and recovery capabilities.

The final effort would produce a full DBMS in Ada complete with several access methods, concurrency control, recovery, an Ada application program interface, query language processor, report writer, bulk load utilities, and application generation tools.

In summary, database technology for single processor systems has reached the state of maturity for the capabilities discussed above and could be provided in Ada for the WIS system by the 1986 timeframe. For more advanced multi-processor, multi-media database management support, a considerable research effort is still required.

2. Functional Requirements

2.1 Introduction

Database management provides the technology for storing, maintaining and controlling access to enormous amounts of diverse data required for large military operations. The key functions provided by this technology are:

1. Data Representation - the ability to represent and store command and control information independent of a specific application, user or program; and

2. Data Manipulation Operations - the ability to reliably perform specific predefined operations on the shared information.

The still evolving database technology is extremely diversified due to the different levels of flexibility and power that are provided. At an extremely simple level, a file system could be used to provide database management functions; the data representation is in terms of a record type and the data operations are simple reads and writes. At the other extreme, a state of the art database management system could support complex data representations and powerful data manipulation operations. In general, there is a trade-off between the development costs and the power and flexibility of the data representations and operations provided by a database management system. The objective of this section is to give an overview of data management technology in order to understand these trade-offs.

In Section 2.2, we describe the general database management functions that are required independent of representation and operator power of the Database Management System (DBMS). Then, in Section 2.3 we describe the levels and alternatives for supporting database management functions. Next, Section 2.4 discusses performance metrics for those functions. Finally, Section 2.5 discusses the key issues that must be addressed in developing a DBMS for Ada.

2.2 General Database Management Functions

Database Management capabilities and the interfaces through which

Capabilities

- Database definition.
- Retrieve and update (store, modify, and delete) data.
- Authorization control.
- Multiple user updates.
- Back-up and recovery.

Database Interfaces

- Programming language interface.
- Query language interface.
- Screen oriented interfaces.
- Bulk load/unload utility.
- Report writer utility.
- Application generator aids.

Figure 2.1 Database Management Functions

those capabilities are accessed are summarized in Figure 2.1. The capabilities and interfaces are described below.

Capabilities

The Define database capability provides the ability to define the logical and physical structures of the database independent of any one program or application. This data independence allows the programs and applications to evolve in response to changing development and mission requirements. The logical database definition defines how information is to be represented in the database. This could include the record types, data fields, relationships between record types, and integrity constraints. The separation of these definitions from the programs allows for record types and fields to be added or deleted from the database definition without changing the programs that do not need to access or update the additional information. The physical database definition defines the file organizations and access structures that will be used to contain the logical records. In a state-of-the-art DBMS, the separation of the physical definition from the logical definition allows access methods to be added or deleted in order to improve performance without changing any of the application programs.

The retrieve and update capabilities are used to retrieve and update information in the database. State of the art DBMS's allow these operations to be performed by logically specifying what data is to be retrieved or updated rather than how the data is to be located. The user specifies the properties of the data such as keys or values of the fields; the DBMS then selects and uses any advantageous access paths. It should also be possible to retrieve and update sets of records with one command as well as retrieve and update individual records.

The authorization control capability permits a Database Administrator (DBA) to specify which data can be accessed by which users for which kinds of operations.

The multiple user capability maintains the consistency of the database while it is being updated by multiple users. The DBMS should prevent users from reading portions of data which are partially updated or writing data that another user is reading.

The back-up and recovery capability is used to recover the database in cases of software and hardware crashes. The back-up capability should support the saving of a copy of the database and the changes that have been made to the database since the copy was made. The recovery capability should restore the database from the copy and then apply the changes.

Database Interfaces

A Programming Language Interface allows programs written in high level programming languages, such as Ada, to retrieve, add, modify and delete data from the database. The specifications of which records to retrieve or update can be based on keys or arbitrary properties of the data, or on a specification of how to find the data in the database. The programming language interfaces to a state of the art DBMS should allow both both record at a time and set at a time processing. These interfaces will be used by application programmers and should thus provide the most flexibility and power in accessing the databases.

An Interactive Query Language Interface allows users at terminals to retrieve and update data in the database by specifying the records or sets of data to be retrieved or updated. These interfaces are generally used by programmers and experienced database users and still provide considerable flexibility in manipulating data although less than is normally possible through the programming language interface.

A Screen Oriented Interface allows a user to select data for retrieval and update through menu selection and form completion. These interfaces can be used by casual and ad-hoc users but generally provide much less power than the other types of interfaces.

A Bulk Load/Unload Utility is used to exchange large quantities of uniformly structured data between the database and files. The interfaces are required for loading large amounts of data from sensors or for preparing data for input into other programs such as spreadsheets, text processors, etc.

A Report Writer is used to specify the format of printed or screen oriented reports and generally provide for automatic headers and footers, page numeration, and data subtotals and totals.

An Application Generator is used by application programmers to specify the user interaction and the effects of the user interactions on the database. The specification usually includes the definition of a form and menus, where values on the form are to come from (the user or the database), and the actions that are to take place as the user completes fields on the form or makes menu selections. The application generators are designed to provide uniform styles of interfaces to the users, and greatly reduce the costs of application development and modification.

In summary, database management technology incorporates a wide range of capabilities that are available through a wide range of interfaces. In the next section, we describe software levels that are required to support the database functions.

2.3 Architecture Levels and Alternatives

The software architecture layers required to support the database

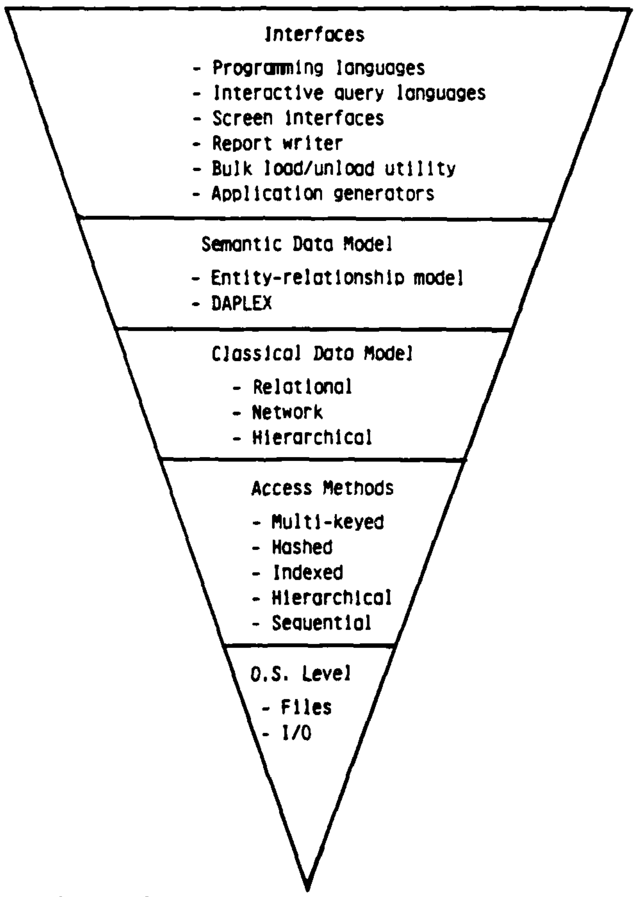


Figure 2.2 Database Management Levels

management technology are shown in Figure 2.2 Each of these areas is discussed below.

Operating System Requirements

The operating system level is the bottom-most layer of the architecture. This level of DBMS software provides basic file access, buffer management and interprocess communications services. These services are similar to the services provided by an operating system but are tailored specifically to the DBMS. Ideally, most of the services are provided directly by the underlying operating system and there is very little DBMS code at this level. The extent to which this can be achieved often has a large impact on overall DBMS performance. A summary of requirements is presented below.

In the area of file access, the requirements are control over the physical allocation of pages on secondary storage and control over when and in which order pages are written back to secondary storage. To improve efficiency, modern operating systems keep a cache of recently referenced file pages in main memory. Buffer management requirements are concerned with the way in which pages are replaced in this cache. In most operating systems, a least recently used (LRU) replacement algorithm is used to replace pages when the cache fills up. This type of replacement algorithm is not well suited for the type of processing done by a DBMS. A DBMS often knows the likelihood that a page will be reaccessed. To efficiently utilize the cache, the operating system must allow the DBMS to pass this knowledge to its buffer manager. The final requirement is interprocess communication. For security, efficiency and flexibility reasons, it is desirable to have DBMS applications running in separate operating system processes from the DBMS. To achieve this, the operating system must provide an interprocess communications mechanism that is oriented towards passing potentially large volumes of data between the DBMS and the application program.

To achieve machine independence, it would be desirable to make the above services available through a standard Ada runtime system. Note that the Ada tasking mechanism, which is a program relative concept, would not provide the required interprocess communications facility since the DBMS and its applications would be running as separate Ada main programs.

Access Methods

An access method is built on top of a host file system and is generally used to provide update and retrieval access to one type of record in a file. Given a ´key´ or property of a record (or set of records), the access method typically updates or retrieves the record (or set or records) with the given property. The most common kind of access methods are described below.

A sequential access method accesses (retrieves or updates) the records in a predetermined order. The properties of the records must specify either the first record or the ´next´ record in a sequence. For fixed length records, the access method may support accessing the ´nth´ record.

A hierarchical access method accesses several record types in a predetermined hierarchical order. For example, a department record access would be followed by accessing all of the employees in a department. As with a sequential access method, the properties of the record are positional in nature, i.e. the first department record, the first employee record for that department, the next employee record, the next department record, etc.

An indexed access method can be used to access records in the file with a given key (not necessarily unique) value. An index tree can be maintained and used to locate the records with a given key value. The common index access methods such as ISAM and VSAM often support both indexed and sequential access for a file.

A hashed access method is used to provide rapid random access to a specific record or set of records. A hashing function is used to compute a potential disk address for a given key value. The record(s) will be stored at (or 'near') that address. A hash based access method generally provides the fastest access method for a single record but does not efficiently support the sequential retrieval of all of the records.

A Multi-keyed access method provides for the accessing of records according to the values of more than one key. For example, one could access employee records by social security number or by date hired. Multi-keyed access methods are generally implemented by combinations of the other access methods. In some multi-keyed systems it is possible to specify primary and secondary indexes.

The access methods above and certain variations have been used to implement what is sometimes called a File Management System (FMS). An FMS supports some of the capabilities of a more general DBMS and can be used by themselves from programming languages and some interactive query languages. As the FMS grows in complexity and power to support shared access, they begin to resemble a full DBMS.

Classical Data Models

A state of the art DBMS can be built on top of access methods and file systems like those described above. The DBMS, however, will typically support data representations of information contained in multiple types of records. The data representation power of commercial DBMS's can normally be described in terms of one of the hierarchical, network or relational data models. In addition to providing different data representation and structuring capabilities, current systems often differ widely in the power of the supported operations. The data representation and operation capabilities normally associated with the three classical data models are described below.

The hierarchical data model in fact was originally based on hierarchical access methods which contained multiple types of records arranged in a predetermined hierarchical order. A full DBMS which supports the hierarchical data model today may in fact support multiple hierarchies and use a variety of access methods to implement the hierarchies. Thus from a data representation point of view, a hierarchy represents a logical relationship between the record types and not necessarily how those records are stored on disk.

The network data model was defined by the CODASYL Data Base Task Group (DBTG). The DBTG model was a generalization of the hierarchical model in that it allowed a record to be part of more than one hierarchy. The DBTG network model definition originally included only record at a time processing commands such as find 'first record' or 'find next' record operations. A DBMS supporting the DBTG network model could be built on top of any of the access methods discussed above.

The relational data model was defined as a collection of independent record types called relations. Fields in these record types are called attributes. Relationships between different relations can only be based on field values in the relation. It is the simplicity of the data structures which has facilitated the development and implementation of the powerful set of operations generally associated with relational query languages. The relational data model, like the other two data models, could be built on top of combinations of the access methods. It would even be possible to utilize a hierarchical access structure.

While the three data models could be built on top of a library of access methods, they historically have not been. The DBMS's instead implement their own access methods in order to have more direct control over the concurrency, security, and recovery mechanisms.

Semantic Data Models

The next level of database management technology will be provided through database management systems that support a 'semantic' data model. A semantic data model generally encapsulates powerful data representation constructs such as those found in the network model together with the powerful database operations normally found only in relational systems.

The best known semantic data model is probably the entity-relationship model. That model is used to represent information about objects or entities in an organization and the relationships between those entities. The relationships that can be specified are more general than those that are possible with the DBTG network model.

Currently, commercial systems are being extended to include the power and flexibility found in what we are calling semantic data models. Commercial DBTG vendors are providing or developing a relational query capability. Similarly, relational vendors are beginning to support referential integrity where only the legal field or attribute values of one relation must match the values of keys in another. Both of these developments are borrowing heavily on the research and implementation experience of the alternate database models.

Interface Level

The final level of database management technology is the interface or applications level. These interfaces described in section 2.2 can be used to invoke database management functions on operating system files, file management systems which support one or more access paths, DBMS's that support one of the classical data models, or one of the semantic data models of the future. In general, the higher the level of the interface to the data management functions the easier it should be to develop, maintain and modify the applications. It should thus be more cost effective to develop an application using any one of the three classical data models than if only file systems were used. On the other hand, the development costs of a general purpose DBMS can be expected to be much greater than that of a file system. Furthermore there is often a performance overhead associated with going through several layers of software.

One of the key design decisions at this layer is how to best make the powerful database management capabilities available to the different classes of users. The same level of database operations should be available through any of the different interfaces. The programming language interfaces should be able to invoke the full power of the query languages. Similarly, the report writers and bulk unloaders should be capable of operating on any subset of data that can be specified through a screen or interactive query language. It is through the careful integration of these capabilities that the full power of the DBMS's becomes most useful. In most current commercial DBMS's the required integration is lacking.

2.4 Database Performance

The performance that can be expected by a DBMS is extremely difficult to quantify and predict. The performance is extremely dependent on the specific application and the computer systems on which the system is running.

One metric for database performance is in terms of the number of update operations which can be performed in one second. On the largest mainframe systems, for the simplest applications, DBMS's have been reported to achieve 10 to 20 updates per second. For the super-mini class machines, updates rates have normally been limited to 2 to 8 updates per second. For 16 bit micro processors, these rates are probably down to 1 or 2 updates per second. Unfortunately, these times are for extremely simple updates which update one record based on one access path. Typically, a minimum of three disk I/O's would be required for each update. For real applications, factors such as the sizes of the records, sizes of the databases and the number and types of access paths will greatly effect the system performance.

Other metrics that have been used for DBMS performance can be based on the time the DBMS takes to retrieve a 'large' set of records in response to one command. Unfortunately, this metric is even more application dependent and general figures such as records per second are generally not very meaningful. For a given application, however, values for this metric could be estimated for different DBMS's.

In general, the performance challenge for an Ada DBMS is to provide considerable flexibility in choosing and combining access paths so that the physical database design can be tuned for a given application. Even with this flexibility the database performance requirements may have to dictate the class of machine which is suitable for a given application.

2.5 Technical Challenges

In summary, the development of database management technology must provide for the efficient use of varying levels of capabilities and permit continuing evolution for increased functionality. The key technical issues that must be addressed for an Ada compatible system are:

1. Supporting data management functions in the Ada environment

2. Accessing those functions through Ada Programs

Each of these issues is discussed in turn.

Database Management for the Ada Environment

There are three basic approaches to providing database functions in the Ada run time environment. The first approach would be to import one or more existing database management systems. The advantage of this approach is that an available and stable DBMS could be made available in the very near future. However, there are some very important issues that must be addressed in any such importation. Consider the two ways in which foreign DBMS software could be introduced into an Ada runtime environment. The first method results in a tight coupling by using the Ada pragma "interface" to load the DBMS software together with an Ada application program. In all commercial Ada implementations that we have examined to date, this method would fail because of conflicts between the DBMS software and Ada runtime system software's use of operating system resources. For example, if the DBMS software is running within an Ada task any I/O operations done by the DBMS software will most likely cause the entire Ada program to block because the Ada runtime task scheduler is unaware that the DBMS software has issued an I/O request. Perhaps a better method is to loosely couple a foreign DBMS with an Ada runtime environment. In this method, the DBMS software runs as an independent operating system process. Ada application programs communicate with the DBMS by making Interprocess Communication calls (IPCs) that are supported by the operating system. This method would require that such an IPC interface be supported by the Ada runtime system.

A second approach to providing DBMS capabilities within the Ada runtime environment would be to develop an integrated DBMS in Ada. This approach would result in a more portable DBMS and applications that use the DBMS. However, some of the issues for the imported system still hold for a DBMS developed in Ada. Because the DBMS software is the manager of a shared database resource, should it be viewed as an indepedently executing program or should it be loaded as a common task with respect to all of the Ada application programs. Implementing the DBMS in Ada solves the problem of conflicts with the Ada runtime system (assuming the low level capabilities required by the DBMS are provided by the Ada runtime system). However, the requirement to provide service for independently executing application programs would probably dictate that the DBMS itself run as an independent Ada program. Again, this would require that the Ada runtime system provide an efficient IPC mechanism.

A third and more general approach would be to build a library of database management components. The library could include separate access methods, support for one or more of the data models, separate utilities, such as bulk loaders, report writers, etc. One advantage of this approach is that a subset of components, such as an access method, could be used by application programs that do not require full DBMS capabilities. In addition, full DBMS programs that are tailored for particular environments could be constructed by selecting an appropriate set of components. The challenges of this approach would be to coordinate the support for multiple users, recovery, authorization, buffer management and other functions that would be commonly shared in an integrated DBMS. A key issue that must be addressed is whether these components can be implemented as Ada generic packages or whether they must be description driven. If Ada generic packages are used, a DBMS component would be instantiated for a particular database definition. The access methods, for example, would be compiled against the specific record types in the database definition. This approach may offer performance gains at the expense of functionality (i.e. very limited multi-user capability, fixed size fields and records). If description driven components are used, a single DBMS component is used for all database definitions. The access methods, for example, would store data as streams of bits and would use the record types in the database definition to interpret this data. Unchecked conversions would be used to move the data into Ada records defined in the user's application program. This approach offers increased flexibility at some performance cost.

Ada Programming Language Interface

In addition to making the database management technology available in the Ada environment, the relationships between the database languages and Ada must be addressed. The Ada programming language interface to the DBMS capabilities can also be provided through several alternative approaches. First, each Ada program could generate the character string which composes the DBMS command and send that string to the DBMS. The advantage of this approach is that it could be used to access any DBMS that could accept strings of commands from other processes. There are three main disadvantages to this approach. First, the specific DBMS syntax is embedded in the data of the application programs. This would make portability between different DBMS's (even if they supported the same data model) practically impossible. A second disadvantage would be that the Ada program would have to use string manipulation and unchecked conversions to build the syntax of each command. This can be a very error prone and time consuming process. Finally, this approach would not allow for any compile time optimization of the database requests.

A second approach would be to develop a Database Package Generator. The application programmer would use a utility to specify an Ada compatible visible portion and a parameterized database command. The visible portion would contain Ada record definitions and entry points to intialize the command and to fetch records one at a time from the DBMS. The database command would embed the parameters passed through the initialization entry point into a general database language. The utility would submit the database command to the DBMS for optimization and generate an Ada package that could be linked into a user Ada program. The Ada program would supply the required parameters and invoke the database command. The Ada package that was generated by the Data Package Generator

Utility would pass the command to the DBMS and buffer the records to be returned to the user Ada program. That program would retrieve one record at a time through a fetch record entry point. This approach could be used to achieve a degree of portability by specifying the database commands in a generic language which could be translated to the syntaxes of different systems. One disadvantage of this approach is that the applications programmer must go through two steps to write programs that access a database. A second disadvantage is that the Ada program which invokes the generated database package, must invoke the package entry points in the correct sequence.

These disadvantages motivate the third approach where the database commands are integrated with the Ada commands. A preprocessor generates the Ada packages similar to those described above. In addition, the preprocessor generates the correct Ada program which calls those packages. This approach is taken by many commercial DBMS's to provide programming language access to databases. It is the easiest for the application programmer to use, allows optimization of the database operations at preprocessing time, and allow for the development of alternate preprocessors to provide portability.

3. Case Studies

As outlined in section 2.3, the database functionality can be provided at several different levels and through several alternative data models. Futhermore, it is difficult to estimate the costs of developing any one of the levels or alternatives since database management system development is an evolving process and in some sense is never completed. Historically, many of the existing systems started as research prototypes and later evolved into commercial products. These and other DBMS products normally undergo tremendous revisions once they have been released to both improve performance and add functionality. Thus the time and level of effort required to develop a given system is not necessarily a good measure of how long it would take to implement that same system again.

This section presents case studies of four database management systems. The systems studied were selected based on the amount of detailed performance and development data we had available. In fact that was our primary motivation in selecting two systems that were developed by CCA. For the other systems, it should also be pointed out that our development estimates are based on personal, informal contacts and do not represent estimates from the vendors themselves. Similarly, the performance results are not based on our own benchmarks and should not be used to make any comparisons or absolute conclusions about the systems performance. Instead, the performance figures are representative of the types of performance that can be expected on a given class of machine for a given application. Case studies are included for the following systems:

1. Model 204 - is CCA's commercial system with relational capabilities that is available on IBM systems. It has been included because a detailed estimate of the level of effort required to reimplement the system in a high level language has been developed.

2. Oracle - is a commercially available relational system that runs on 32 bit mini computers. It is included since it was one of the first relational systems and has been ported to a number of different systems.

3. DBMS-20 - is a commercially available DBTG network system and is included in order to compare it to the relational systems.

4. LDM - is a prototype system being implemented (95% complete) in Ada by CCA. It is included because of the relevant Ada experience and because it provides us with detailed and comparable size estimates for the various components of a DBMS.

These systems are described below.

Name of System: Model 204.

Developer: Computer Corp. of America, 4 Cambridge Center, Cambridge, MA 02142.

Description of System: Model 204 is a general purpose database management system with relational capabilities.

Database Definition: A Model 204 database is made up of one or more files. Each file consists of a set of fields. Records in the file can be kept in entry order, sorted, or hashed on a single key field. In addition a hashed based index on any field can be requested

Retrieval and Update Capabilities: Retrieval and update functions may be performed on individual records or sets of records. Data selection is based on field values and correlation of field values.

Authorization Control: Security in Model 204 is password driven. Authorization can be specified at the levels of login, file, procedure, record, field, and terminal.

Multi-user Capability: Model 204 is designed to operate as a single, online nucleus supporting a large number of simultaneous users with true multi-threading.

Backup and Recovery Capability: A dump/restore utility and a checkpoint/restart rollback/rollforward utility is provided.

Interfaces: Model 204 supports programming language subroutine call interfaces for COBOL, FORTRAN, PL/1 and assembly language; an interactive query language interface with relational power; screen oriented application development aids for IBM 3270 type CRT's; a report writer; and bulk load utilities.

Performance: In one benchmark study using a dedicated IBM 4341 group 1 with 8 megabytes of main memory and running VM/SP, with Release 6.3 of Model 204 running a VS1 guest operating system, the insertion of 500 records, each with 10 indexed fields and 50 non-indexed fields and committed individually, took 115 seconds. This translates to about 4 updates per second.

Quality: Model 204 is a production quality DBMS.

Development Team: Model 204 has continually evolved over the last 15 years. Information on its development over this period is not considered to be a reliable estimate of the effort it would take to develop Model 204. Instead, we include the estimate of 18 labor years to rewrite Model 204, without any architectural changes, in a high level language. Note that this estimate does not include the cost of system design and documentation.

Development Environment: Model 204 currently consists of some 150,000 IBM 360/370 Assembly language statements comprising some 1500 subroutines in some 70 modules.

End Use: The first Model 204 system was sold to the Defense Department in 1971. Today, there are approximately 200 installations of Model 204 running Release 6.5. Model 204's diverse applications range from CAD/CAM applications at McDonnell Douglas, to budgeting applications at the White House. The size of user organizations range from several thousand programmers at ATT long lines, to a handful of programmers at the Boulder Valley Public School System.

<u>Comments</u>: Model 204 is relevant for this study because it is quite widely used within several Defense Department agencies. Its principal strengths are its integrated interface and its high level of performance. However, Model 204 achieves its high performance only by implementing its own proprietory access methods at the channel program level. Thus, the system is not easily transportable.

Name of System: Oracle.

Developer: Oracle Corp., 3000 Sand Hill Road, Building 3-180, Menlo Park, CA 94205.

Description of System: Oracle is the first commercially available relational database management system. It supports SQL, the relational language originally developed at IBM Research.

Database Definition: An Oracle database consists of a collection of relations. Any field or attribute in the relation may be indexed with a compressed B+ trees (similar to VSAM) index.

Retrieval and Update Capabilities: The data manipulation facilities of SQL provide complete physical data independence. Retrieval and update functions may be performed on individual tuples or sets of tuples. Data selection is based on field values and dynamically defined relationships (through matching among field values).

Authorization Control: Each relation in an Oracle database is owned by its creator. Grant and Revoke commands are used for the delegation and revocation of access privileges to other users.

Multi-user Capability: Oracle supports multiple concurrent batch and online terminal updates and queries to the database. Concurrent accesses are synchronized by locking at the relation or physical page level.

Backup and Recovery Capability: Checkpoint/restart, rollback/rollforward facilities are provided for recovering from transaction failure, system failure, and media failure.

Interfaces: Oracle supports a relational query language called SQL. All SQL facilities can be used directly from a terminal or embedded in programming languages like COBOL, FORTRAN, BASIC. In addition to the basic SQL facilities, an interactive application generation facility and report writer are supported.

Performance: According to one study using Oracle Release 2.3.1 running on a dedicated VAX 11/780 with 4 megabytes of main memory, the following benchmarks were obtained. With 8 concurrent users updating one field in one tuple of the the same relation based on a unique key field took 8.69 seconds of real time on the average. This translates into about one update per second. In a similar benchmark with only 4 concurrent users, about 2 updates per second were possible. Note it has since been claimed that later versions of Oracle have significantly improved performance. However, the benchmarks still illustrate how factors such as number of simulataneous users can greatly affect the performance of a DBMS.

Quality: Oracle is commercially available. However, the version that runs on IBM machines is still being operated in beta test mode.

Development Team: According to our sources, the development of the first releasable version of Oracle took 11.25 labor years over a 2.25 year period. This included a 5 person assembly language effort for 1.5 years and a 5 person effort over 9 months to convert and extend the system in

C. For the past 2 to 3 years from 5 to 10 people have been involved with maintainence and further development. Reports have indicated significant performance and functionality improvements over the originally released version. We thus feel that the originally released version was closer to what we would call an operational prototype system.

Development Environment: Oracle currently consists of about 2000 modules totalling some 250,000 lines of C code. It is designed to operate on almost all machines that support a C compiler. Transporting to a new operating system environment requires changing about 30 of the modules. Oracle currently runs on DEC PDP-11 under RSX-11M+, RSX-11M, IAS, RSTS, and UNIX; on DEC VAX-11 under VMS and UNIX; and on IBM 370 series, 3000 series, and 4300 series under VM/CMS and MVS.

End Use: The first Oracle system was installed in June 1979. Today, there are approximately 250 commercial and government installations. Typical application areas include finance and accounting, manufacturing, marketing and sales, and security analysis.

Comments:

Oracle is the most portable relational database system today because of the wide availability of C compilers.

Name of System: DBMS 10/-20.

Developer: Digital Equipment Corporation, 146 Main Street, Maynard, MA 01754.

Description of System: DBMS 10/-20 are specifically designed to work within the environment of DECsystem 10/-20 under the TOPS 10/-20 operating systems. They implement the 1971 CODASYL DBTG network data model.

Database Definition: Each database is made up of a collection of record types, set types, and areas. Each set consists of an owner record and a number of member records and represents a one-to-many relationship between the owner and the members. Depending on access requirements, embedded pointers to owner, next member, prior member can be maintained. Multiple access paths to a record are provided by allowing a record to belong to multiple sets of different types. Records can be placed within an area by hashing, relative addressing, or close to the owner of a set occurrence of which it is a member. This is similar to a hierarchical access method.

Retrieval and Update Capabilities: Retrieval and update functions can be performed on individual records only. A collection of 17 procedural verbs are provided for navigation within the network database. These include operations for opening and closing areas, storing and deleting records, inserting and removing records to and from a set, and six different forms of finding a desired record.

Authorization Control: This is handled through privacy lock and key mechanisms.

Multi-user Capability: Mutual interference between transactions is avoided through locking. Access control statements are included in the data definition to specify whether a running programming is to lock each area for the duration of a transaction, or to lock only those pages that are actually touched by the transaction.

Backup and Recovery Capability: A journal capability is provided to ensure the integrity of updates. The journal file is used to store before and after images of transactions in order to support recovery from both system and media failures.

Interfaces: Application programs can be written in extended COBOL or FORTRAN. The COBOL compiler is modified to accept the extended COBOL. A preprocessor is used to translate the extended FORTRAN into pure FORTRAN with embedded procedural calls on the run time system. No report writer interface, bulk-load interface, or applications generator interface is currently provided. A relational query language interface and a screen-oriented interface are planned for the near future.

Performance: For a 1-MIP machine, up to 20 single-record update transactions can be performed per second.

Quality: DBMS 10/-20 are production quality DBMSs.

Development Team: The original development of DBMS 10/-20 took about 25 labor years The cost of recoding the system in a high level language is estimated at about 10 labor years.

Development Environment: The complete system, including both the COBOL and FORTRAN interfaces, consists of some 175,000 lines of DECsystem 10/-20 macro assembly language statements. These can be broken down into 2 communications modules (2 subroutines), 10 top-level modules (150 subroutines), 6 utility modules (100 subroutines), 4 kernal modules (50 subroutines), and 4 operating system modules (25 subroutines).

End Use: The first installation of DBMS-10 was delivered July 1973. Today, there are over 200 installations of DBMS 10/-20.

Comments: The basic capabilities of DBMS 10/-20 are representative of those found in CODASYL DBDG network systems. The performance of DBMS 10/-20, however, is better than the average network systems because system services provided by the operating system are fully exploited.

Name of System: Local Data Manager (LDM).

Developer: Computer Corp. of America, 4 Cambridge Center, Cambridge, MA 02142.

Description: The LDM is a general purpose DBMS that implements the semantic data model and a data definition and manipulation language called DAPLEX. This is a prototype system that is being implemented in Ada for DARPA and the Navy (NAVELEX) under contract number N00039-82-C-0226.

Database Definition: Logical database definition parallels data type definition in Ada. All data is strongly typed. A database definition consists of a collection of record type definitions (called entity types). Record types may be organized into supertype/subtype hierarchies. Fields may be defined to hold a single value or a set of values. Fields may also be used to define explicit relationships between record types. An optional physical database definition may be included that specifies how a database is to be physically stored. Three types of dynamic file organizations are supported; sequential, random using a linear hashing technique and ordered using a B*-tree. Secondary indexes may be defined to provide multi-key access.

Retrieval and Update Capabilities: Retrieval and update functions may be performed on individual records or sets of records. Data selection is based on field values or relationships. No user knowledge of the underlying physical storage structures is required. The DBMS contains an optimizer that determines the most efficient way to access the requested data.

Authorization Control: Access is controlled down to the field level on a per user basis.

Multi-user Capability: The LDM supports multiple concurrent readers and updaters of a database. Users enter transactions which consist of one or more retrieval or update commands. The LDM transaction manager controls the interleaved execution of these transactions in a manner which insures database consistency.

Backup and Recovery Capability: Backup and recovery facilities are provided that guarantee database consistency across both software and hardware crashes.

Interfaces: An interactive language interface is provided that supports database definition, authorization, retrieval and update commands. A bulk load interface is provided to efficiently load large quantities of formatted data.

Performance: Performance analysis is waiting for a f[illegible] compiler.

Quality: Prototype system - The system is still under development. An initial version, implemented in an Ada subset, using an Ada to Pascal translator, is scheduled for release in March, 1984. A full-Ada version is scheduled for release in January, 1985.

Development Team: The level of effort required to develop the Ada subset prototype system is shown in Figure 3.1. A total of 11.25 labor years is required to produce the prototype. We beleive this is an accurate estimate as this phase of the project is nearing completion. Although this prototype implements the majority of the functional requirements of a modern DBMS only minimal effort has been directed towards testing, tuning and documentation. Large efforts are normally spent in these

Project Phase	Level of Effort (Months)	Elapsed Time (Months)	Staff
Research/ High Level Design	30	15	2
Detailed Design/ Implementation	105	30	3.5

Figure 3.1 LDM Prototype Level of Effort Summary

areas for production quality systems.

Figure 3.2 shows the size of the major areas of the system. These sizes are used in Section 4 to estimate the proportion of time required for developing the different components of a DBMS. Packages refer to the number of Ada packages in each area. Code summaries are given for package specifications and package bodies. These counts refer to source lines that contain at least part of an Ada statment. Blank lines and lines containing only Ada comments are not included. Routines refer to the number of Ada procedures and functions contained in the packages. The User Interface includes the parser, semantic analyzer, query simplifier, terminal interface, output data formatter and various support utilities. The Schema Processor includes facilities for database definition, directory maintenance and directory access. The Logical Database Processor includes query optimization and query processing. The Physical Database Processor includes the operating system interface, buffer management, file management, concurrency control, recovery,

	Packages	Code		Routines
		Spec	Body	
Physical Database Processor	39	5,274	44,102	1,134
Logical Database Processor	12	863	19,527	293
Schema Processor	25	2,311	15,832	387
User Interface	94	7,849	47,182	1,721
	----	------	-------	------
Total	170	16,297	126,643	3,535

Figure 3.2 LDM Prototype Code Summary

record management and access methods. To give a feel for the amount of code required to implement the access methods supported by the LDM, a further breakdown of the Physical Database Processor is presented in

	Code	Percent of Physical Database Processor
Access Manager Utilities	12,778	29%
Access Managers		
Sequential	2,474	6%
B-Tree	5,815	13%
Linear Hashing	3,548	8%
Concurrency control, Recovery, Buffer Manager, File Manager	11,400	25%
Semantic Data Model specific support	8,087	18%

Figure 3.3 Physical Database Processor Code Summary

Figure 3.3.

4. Analysis

This section presents an approach for providing near term database management capabilities within an Ada runtime environment and outlines capabilities that are expected to be available in the long term. Section 4.1 gives an estimate of the cost of quickly providing an iterim capability by interfacing to an existing (non-Ada) DBMS. Section 4.2 gives an estimate of the cost of implementing a DBMS in Ada. Finally, Section 4.3 presents recommendations for mid to long term capabilities that should be considered. Note that no attempt has been made to compare or recommend existing relational systems. The selection of a system should be based on the functional and performance requirements of command and control applications. Such a requirements analysis is beyond the scope of this report.

4.1 Interm Capability

An interim database management capability can be achieved rapidly within an Ada runtime environment by importing an existing relational DBMS. It is recommended that the loosely coupled method, described in section 2.5, be used to achieve the connection. The recommended user interface would be the Database Package Generator described in section 2.5. The Ada application programmer would specify a parameterized set of database operations. The Database Package Generator would then create an Ada package that contained the calls on the DBMS to implement those database operations. The estimated cost of this interim capability is 1.5 labor years over an elasped time of 9 months. This assumes an existing DBMS on the target system and interprocess communication capability (IPC) provided through the Ada runtime system.

4.2 Near Term Capability

The recommended near term capability is an easily reconfigurable database management system implemented in Ada. This system would be built out of a library of components. This approach would have several advantages. First, components (such as access methods) themselves could be used in application programs that do not require the full power of a general purpose DBMS. Second, DBMS configurations tailored to specific applications could be supported by selecting appropriate access methods or user interfaces. As discussed in section 2.5, one open issue is which components can be implemented as generic packages and which ones must be description driven. Our feeling is that this issue will not have much impact on the overall cost of providing the DBMS capability. It is unlikely that an existing DBMS can be simply recoded in Ada and still achieve the desired degree of modularity required to form a good library of components. A careful design of the interfaces between these components must be conducted to insure that they can be used alone and in combination without significant impact on performance. It is our belief that such an analysis would result in architectural changes to most systems that would prohibit a simple *translation of existing* code into Ada.

The estimated cost of a near term DBMS capability is shown in Figure 4.1. These estimates are based on information obtained from the case studies and CCA's own experience in implementing database management systems including implementing a DBMS in Ada. Estimates are given to reach both the operational prototype and production quality stages. The PDP subsystem includes low level components simiiiar to the ones shown in Figure 3.3. The full system includes several access methods, concurrency control, recovery, Ada application program interface, query language processor, report writer, applications generator and bulk loader.

	Prototype Level of Effort (Years)	Production Level of Effort (Years)	Elapsed Time (Years)
PDP Subsystem	4	8	1.5
Full System	12	25	3.0

Figure 4.1 Cost Estimates for Ada-based DBMS Capability

4.3 Mid and Long Term Capabilities

In order to meet the mid and long term requirements for command and control, advances in database technology in the following areas will be required.

Distributed and replicated database capabilities will be required for performance and survivability. The size of the totality of command and control information necessitates that the database be distributed across several processors. Furthermore, critical data should be replicated on geographically dispersed processors in order to insure that it will be available when needed. The applications should be built independantly from the distribution and replication of the data. The DBMS should provide for the automatic location of the required data and the consistent updates of the replicated copies. Otherwise programs would have to be changed whenever data is moved or copied. There are currently no commercial DBMS's available that provide for automatic distribution and replication, although several prototypes have been developed. It is expected that this capability will be available for WWMCCS in the next 3 to 4 years.

Real time DBMS capabilities may be required in order to provide rapid enough response time. A real-time DBMS system would provide for separate database definitions, logical retrieval and updates independent of access paths, authorization, multiple user, back-up and recovery capabilities for data maintained in main memorey. All of these capabilities would be provided by having programs share portions of main memory. Existing DBMS's, on the other hand, have not been designed to manage data in main memory. A research effort would be required to develop this technology for real-time command and control.

Multiple Media DBMS capabilities would extend DBMS's to store text, voice, picture, graphics, and geometric data together with formatted records. Currently, separate systems are used for each of the current types of data. With this approach, it is often difficult to query the different types of data. In addition, security, multiple users, and back-up and recovery capabilities are either not existent with the other kinds of data or must be separately implemented. Again a research effort would be required to sufficently develop this capability for long range command and control systems.

Expert System Support capabilities will be required to implement expert systems on top of large shared databases. Existing expert systems are built on top of small databases which in many cases are entirely contained in main memorey. Thus the information used by the expert systems is generally not directly available in a shared, separately defined database. Similarly, existing DBMS's do not directly support the specifications of rules about the data they maintain. The evaluation of the rules then can not be directly integrated with the DBMS query optimization and processing. As a result the existing expert systems technology is not readily extendable to large quantities of data. Again a research effort would be required to extend database technology to support and enhance expert systems.

5. Conclusion

In conclusion, database management technology will play an evolving role in the development and support of WWMCCS. Database Management Systems will be used to reliably store critical information, adapt to changing uses and content of that information, and eventually may provide real-time, selective and intelligent access to the specific critical information that is required for a given situation.

The database technology today has reached a state of development so that an adaptable, single site Database Management System can be developed in Ada within a 3 year time period with a 25 labor year level effort. The system could be developed to provide both integrated DBMS capabilities and to allow the construction of tailored DBMS capabilities to meet the needs of a given application. The key technical challenges for that effort will be to integrate those capabilities with the Ada run time environment and the Ada programming language.

In the interim period it would be possible to develop Ada interfaces to existing relational DBMS's. It is estimated that this effort would require a 1.5 labor year effort over a period of 9 months. Again, the challenges of this effort would be to support the foreign DBMS as a separately running process and to readily access the data from an Ada program.

Distributed database management technology is maturing as several prototype system development efforts are underway. Production quality systems are expected to be available in 3 to 4 years. The WWMCCS development should plan to incorporate this technology at that time.

In the long term, a considerable research and development effort would be required to provide a multi-processor DBMS which supports multiple types of data including text, image, graphics and voice data and provides the rapid real-time and expert system responses that could be effectively used in command and control.

TEXT PROCESSING SYSTEMS

Technical Area Report

Prepared by

Newburyport Computer Associates, Inc.
27 Fair Street
Newburyport, Massachusetts 01950

for

INTERMETRICS
Washington Division
4733 Bethesda Ave. Suite 415
Bethesda, Maryland 20814

1. OVERVIEW

The product under discussion is a text processing system which in its simplest form is a word processor and in its fully-implemented form comprises the text-based components of a full electronic office system. The most basic system would provide full text editing of documents of any size, on-screen formatting in real time, and background output to a letter-quality printer. It would have a fully interactive word processing-style interface. The more advanced version would support printers with multiple-font and size capability and typesetters, and would include features such as teleconferencing, electronic mail, and integrated graphics. It might have a more sophisticated user interface (for example icons and a mouse) and would allow for rapid context switching between environments as well as simultaneous viewing of different types of user files.

In order to distinguish the basic level system from the more advanced system, we shall use the term "word processor" to mean a basic-level system, and "text processor" to mean a more advanced office automation system.

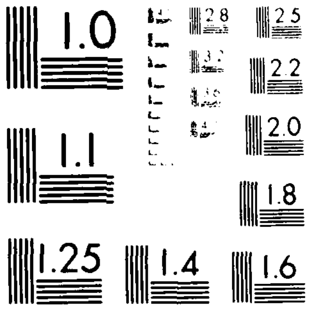
The case studies describe two word processors and a collection of various aspects of some typesetting projects. The two word processors run on standalone microcomputers under single-user operating systems. One was written for a European company as a dedicated word processing product in assembly language, the other was written in UCSD Pascal in the U.S. as a software package to run on a manufacturer's machine. The first was designed and largely written by the authors, the second was completely written by the authors.

The systems which drive typesetters are discussed in order to provide some basis for the analysis of more advanced features especially as related to level of effort estimates.

AD-A142 570 WIS IMPLEMENTATION STUDY REPORT VOLUME 3 BACKGROUND INFORMATION(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T H PROBERT 01 OCT 83 IDA-D-51-VOL-3

3/6

UNCLASSIFIED IDA/HQ-84-28344 MDA903-79-C-0018 F/G 17/2 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Performance Metrics

Performance is critical in word and text processors because of their highly interactive nature. A good word processor has real-time on-screen formatting, immediate response, and a sophisticated user interface. In fact, the main distinction between a true word processor and a text editor lies in the user interface more than in the functionality, although there certainly are functional differences as well. Word processors have become ubiquitous not only because they are vastly more efficent than typewriters but also because they are easier to use than standard computer text editors. This ease-of-use is a result of not just the improved style of the user interface but also of superior performance.

Specific performance goals are as follows:

1. Text should be scrollable in the vertical direction at a level of at least 10 lines per second.

2. System response times for the following features should be at the level of a very fast typist (.06 seconds):

   Insert character (including wordwrap down to the next line)

   Delete character (including wordwrap up from the next line)

3. System response times for the following features should keep up with the repeat rate of the keyboard (.06 to .10 seconds):

   Cursor left, right, up or down

   Horizontal scroll of one column

4. System response times for the following features should be perceived as instantaneous by the user (less than .1 seconds):

   Delete word or line

   Cursor jumps to anywhere on the screen

5. System response times for the following features should be less than 1 second:

Reformatting the entire screen (with a 20-line screen this is 20 lines per second, with a full-page screen it is 66 lines per second)

Move or copy blocks of a few lines.

Level of Effort of Implementation

As shown in the case studies, a basic word processor requires about 3.25 man-years of development time in a higher-level language, assuming a development team consisting of senior level personnel. The assembly language project described in the case studies required about the same level of effort, and included development of a file system to support scrolling in both directions of the word processing files and a simple dedicated operating system. The two systems are roughly equivalent in functionality. However, an important difference between the two systems is the quality of the user interface. The higher-level language word processor has a vastly superior user interface, and in addition supports multiple kinds of terminals and printers.

These experiences show that an elapsed time of 18 months from conception to product should be assumed as a minimum for a word processor. At least three months should be allowed for in-house quality assurance and beta testing at user sites.

The additional effort required for each capability beyond the word processor stage can be estimated by examining individual projects which implemented certain of the features that would be included in an advanced text processor. The following project required 6 man-months of one senior-level person with extensive experience in that application:

Batch (non-interactive) typesetting hyphenation and justification (formatting) program using an average development environment in assembly language on a single-user minicomputer.

The following project required appproximately 2.4 man-years using middle level people with some experience in the area:

An interactive justification package for a dedicated direct-entry, micro-based typesetter, without hyphenation. This project was developed in Assembler using a primitive microcomputer development environment.

The following project required about 10 man-months of one senior person and 3 man-months of one junior-level person:

A three-user micro-based 'word processing' and typesetting system with a letter-quality printer and a small typesetter online. The effort involved the development of a simple editor, batch hyphenation and justification, and typesetter and printer drivers. Development took place on an Intel 8080-based development system in assembly language.

These examples illustrate the difference in development effort between interactive and batch systems, the interactive versions costing considerably more development time. When adding typesetting capability to a word processor, the environment should be kept interactive.

2. DISCUSSION OF FUNCTIONAL REQUIREMENTS

The functional requirements of a word processor can be divided into the following major areas: 1) Editing, 2) Formatting, 3) Printing and Pagination 4) Ancillary support functions. Although several of the functions from the user's viewpoint will overlap two or more areas, most functions fit more logically into one category than another. Besides, this breakdown turns out to be a useful one for allocation of development responsibility as well. The minimum functional requirements for a word processor include the following:

Editing

The word processor should be able to perform all standard editing operations including insert, delete, move, copy, search and replace, named saves and restores. Cursor motion should include not only up, down, left, right, and scrolling vertically and horizontally, but also rapid jumps to text objects such as words, paragraphs, pages. There should be no arbitrary restrictions on document size or the size of move or save blocks. Files should be freely scrollable forwards or backwards.

Formatting

The format of the text (line endings) should be continuously updated during editing, although not to the extent that screen motion becomes so frequent as to be an annoyance to the user. Centering, right-alignment, justification, tabs, and indents should be shown on the screen as closely as possible to the way they will be printed. Some sort of hyphenation aid should be provided, if not an automatic hyphenation. Attributes such as bold, underlining, sub- and superscript should also be shown as accurately as possible. Because output is to a letter-quality printer, the screen should resemble that printer's output as closely as is feasible. Memory-mapped screens are usually the best choice for displaying these characteristics and keeping the formatting changes updated in real time, but terminals can also be used with some restrictions.

Word processors customarily use a format ruler as the vehicle for defining tabs and margins. The ruler should be easily editable. The capability of interspersing multiple format changes (rulers) throughout the text should be available. Formatting should be integrated with editing as much as possible, and it should definitely not require the execution of a separate formatting program. Formatting commands such as center and indent should be entered in a menu-driven or prompted format, contrary to the command formats of most text editors with separate formatters and of almost all typesetting systems.

Formatting is the most critical area because it is both the most difficult part to implement and simultaneously has strict performance requirements.

Printing and Pagination

Some kind of facility for breaking the text into pages and numbering of pages is required, along with the ability to define multiple headers and footers. (Some word processors require the text to be created initially in pages of a fixed maximum length, but that is generally regarded as undesirable by today's standards).

If multiple printer types are to be supported it is desirable to have a general table-driven printer interface. (The same is true of typesetters). Printing should make maximum use of the capabilities of most letter-quality printers which include boldface, underlining, sub- and superscript, changeable printing elements, variable line spacing and variable horizontal spacing for justification. Single sheet feeding and changing of the print element by the operator should be supported in background.

Ancillary Support Functions

An on-line Help facility has become a standard for word processors because of the complexity and number of the functions provided. Having a directory of documents available without leaving the word processor is also extremely desirable.

A mass mailing package that supports simple data entry of names and addresses which can be merged into a template letter and printed should be included a a minimum. Such a package is sometimes expanded to the level of a small data base with sorting and selection of records.

Features for a more advanced word processor could include footnotes and references, a spelling checker, an index and table of contents generator, outline format capability, programmable editing keys, automatic pagination with widow avoidance, equation editing, line drawing, proportional characters, column move operations, column arithmetic, user-definable characters, automatic hyphenation, boilerplate merge, automatic paper handling, multi-column text, undo facility, revision control, automatic abstract generation, vertical justification, hanging indents, pre-printed form fill-out.

To take the functionality to the text processor level would require first level an integration into an office automation system, including an electronic mail and conferencing capability, integration with the system data base (merging of data from the data base into word processing files, access of word processing files by other programs), and integration with other programs such as spreadsheets, project scheduling, note and reminder facilities. Support of full-page screens with characters of multiple sizes and fonts, support of a laser or high-resolution matrix printer and/or typesetter, integrated graphics with diagrams and graphs mergable into the text, and multiple overlapping windows could also be included. With the proper operating system environment, rapid context switching among these programs could be built in. The user interface could be upgraded with the use of icons and an auxiliary pointing device such as a mouse a la the Xerox Star and the Apple Lisa.

Technical Challenges

The single most difficult performance problem is that of keeping the screen correctly formatted in real-time, especially if it is a full-page screen and particularly if it is bit-mapped. Proper design of data structures and fast hardware (CPU and screen) are helpful. More importantly, the number of characters rewritten to the screen must be reduced to an absolute minimum. This optimization, which should be built right into the formatter logic, is particularly essential for bit-mapped screens because of the additional overhead of the character to bit-map translation. Even using a memory-mapped, character screen it is an advantage. On the IBM PC version of CrystalWriter, which uses such a screen, the software optimizes screen character motion. The result is instantaneous insertion and deletion of multi-line blocks of text. Such a feature increases editing efficiency enormously.

Driving terminals can have several problems, including non-transparency of the operating system interface, limited capability especially for attribute display in the terminal itself, too slow a communication line, and inadequate performance of the terminal's internal logic.

Word and text processing files contain a great deal of formatting information, making them very non-standard at least in their internal format. This fact has at least two consequences: 1) transfer of text between the word processor and other programs is not straightforward, and 2) to save disk space the files may need to be compressed necessitating special algorithms to translate between disk and screen formats. These translation algorithms can create as critical a performance issue as screen formatting, in order to meet the vertical scrolling performance goal stated above.

Printer interfaces very often have a similar problem to terminal and screen interfaces, that is, that the operating system intervenes in a manner that makes it difficult to drive these devices at their lowest level. It is extemely important that both the screen and the printer be driven at a low level because of the functional demands of word processing. This requirement is even more true of most typesetters. Most letter-quality printers, matrix printers, laser printers and typesetters come internally programmed with considerable intelligence designed to assist the front-end. It has been our experience that these device-resident ROM programs are mostly inadequate, generally poorly documented, often incorrect or inconsistent, and almost never contain the exact mix of features desired for the particular front-end system being designed. The solution, short of a custom-programmed device, is to have access to the device at the lowest possible level.

CASE STUDY 1: Word Processor for Datic Electronica

This product is a dedicated word processor built by Datic Electronica GmBH, Trier, W. Germany. The hardware was based on the 8080 microprocessor, 32K RAM, memory mapped video, dual 8 inch floppy disks and Daisy wheel print mechanism. The product was first installed in May 1977 and has been developed further and made more specialized since then. It is now a recognized leader in some markets in Germany.

The word processor software featured on-screen formatting, virtual scrolling to disk, horizontal width to 132 columns, unlimited number of format changes from 12 user-definable formats, integrated mail merge facility, and background printing. Insert was done by opening the screen at the cursor position, allowing the new text to be entered, then reformatting the text below the cursor. The names, addresses and other text for mass mailings was taken from a record list which was attached to the template document and were referred to by fixed symbols. The merge operation was done during printing and the text reformatted at the same time. Editing was allowed while printing and merging were going on.

There was no support for column operations, spelling checking, proportional characters, communications or data base operations. Horizontal scrolling was done by jumping the screen 52 columns when the cursor reached the edge. This was necessary since the processor was too slow to support continuous horizontal scrolling.

It was necessary to develop a file system, multitasking monitor and a software method to share code.

The performance was acceptable. Vertical scrolling was 6 lines per second and it took about 1 second to close an average screen from insert. At the time this was viable, however there were requests from users that the scrolling speed be increased. In today's market it would be considered a serious competitor in the personal computer word processor market, but not in the dedicated word processor market. This should be viewed as a minimum system.

During the development of this product, there were organizational and personnel problems with several programmers being on the project for short periods. Despite all this, the group worked very hard, averaging over 50 hours per week throughout the course of the project. The effective team consisted of three full time experienced designer/programmers and two junior members who worked on the project about half time. Additionally there was one outside programmer who developed the file system, and a full time manager. Documentation for the system was done separately.

The development environment was quite primitive. MACRO was used, with all development done on the target machines. There were several hardware design problems which in conjunction with slow printers and no software debugger cost a significant amount of time. From the start of the project there was extensive module debugging done, and despite the lack of tools the final product was reliable.

CASE STUDY 2: CrystalWriter

This case study is of a word processor developed in Pascal by the authors at Newburyport Computer Associates, Inc. for Western Digital Corporation's Advanced Systems Division. The development hardware and the initial target machine were the same: the Western Digital Microengine, a micro with UCSD Pascal implemented in microcode. The target printers were both dot matrix and letter-quality types. Typical hardware configurations included two 8-inch floppy disks with an optional 10-megabyte Winchester, 128K of memory approximately 26K of which was occupied by the p-system, and one terminal.

The software was designed to be later upgraded to accomodate proportional characters and bit-mapped screens because the ultimate goal was to generate a more advanced version for a Western Digital machine with a bit-mapped screen that was still under development. Although the system was developed to the point of a market-ready product for the Microengine, Western Digital decided some months later to stop selling the Microengine hardware. The division at Western Digital funding the project merged with another company, which did not choose to pursue development of the ultimate version of the system to run on the more advanced hardware.

The product was developed over a period of about 18 months by a development team of two persons familiar with development of word processors. The total development effort for design, code and test was approximately 3.25 man-years. Quality assurance and development of the documentation were performed at Western Digital over a period of about 2.5 months by one full-time and one part-time person.

The development environment consisted of two Microengines with the UCSD development system, each with two diskettes and a 10M Winchester disk and 128K of memory. The environment was extremely convenient in all aspects except one: debugging. The editor and compiler are better than average for a micro environment, and give the programmer all of the advantages of a coordinated development system. However, we found that the successful debugging of a program of this level of complexity required special purpose debugging software. Once those tools were in place, development proceeded smoothly and rap y. A good symbolic debugger would have saved several man-months of effort.

The development was aided by the availability of the Western ital operating system support staff on the other end of the phone line. Th system was later converted to run on the IBM Personal Computer on which virtually no technical support is available. The dramatic drop in support level was so significant that it resulted in several man-months of wasted effort in the IBM PC version.

CrystalWriter is a basic word processor. It does not meet all of the performance goals cited above because of hardware inadequacies, and it is missing a mail merge facility and horizontal scrolling. Its strengths are in its user interface which is considerably more advanced than the Datic system described as Case Study 1, in its automatic pagination with widow avoidance, and in the on-screen formatting and editing capabilities. It is set up to drive multiple terminals and printers with minimal additional programming effort.

The product was installed at one customer site before Western Digital decided to drop the Microengine from its product line.

The performance was not up to standard for the following reasons:

1) The Microengine is a slow machine by today's standards. For example, typical operations take approximately six times as long to execute on the Microengine as on an Intel 8088. There is too small a time window to perform the more critical algorithms for formatting and scrolling. Most modern microcomputers are fast enough to keep up with the performance goals, although not without careful design. A case in point is the IBM PC version of CrystalWriter. It is written in 8088 assembler with the critical algorithms optimized for high performance. The scrolling algorithm meets the specified performance criteria even including a screen to disk format translation, but there is very little time left over. (The Microengine version does no such translation and still can only scroll at about 4.5 lines per second).

2) The screen on the Microengine is a terminal on a 19k baud line. The performance of the on-screen formatting is limited not by the speed of the transmission line but by the operating system overhead and by the terminal's own logic. In contrast, the IBM PC version drives the memory-mapped screen directly and there is no problem whatsoever with keeping the screen updated.

CASE STUDY 3: Typesetting programs

Quadritek Typesetter

This is a direct entry typesetter developed by Itek Corporation, Rochester, N.Y. The software was done by Bolt Beranek and Newman, Cambridge, Ma. The hardware was based on the National Semiconductor PACE microprocessor, cassette storage, screen and keyboard, and a zoom lens based typesetting mechanism which was the output device. The first shipments were in September 1976 and by 1980 over 12,000 had been sold.

The Quadritek is included in this report because it used a typesetting formatter which was designed for an interactive system. The editing was very limited, but it was not a batch system as most typesetters are. It included mixed point sizes from 4.5 to 36, 4 fonts each 92 characters, extensive tab handling, line lengths to 256 characters, all common justification types.

There was no support for graphics, skews, complex indents (other than single line hanging indents), automatic hyphenation, or page layout. For normal type, the formatter ran about 300 characters per second, very complex type slowed it down by a factor of 2 or 3. The original code was used until it was rewritten for a different CPU in 1981 and was considered a very serious competitor at that time.

The formatter part of the software was done by two persons full time and one person about half time. All were moderately experienced and worked full time but without excessive overtime.

The program was done in Assembly language on a host processor. There was no software debugger which wasted a significant amount of time. Extensive code reviews were done and the project was done first as a prototype, then rewritten and expanded into the final product. The result was very reliable.

Graphics Arts Terminal

This project was to develop a combination word processing and typesetting system for three users based on a dual-8080 floppy-based system packaged with shared memory by Terminal Commmunciations, Inc. of Raleigh, N.C. The product was eventually marketed by Telex as the model 2100. The product was shown at DRUPA in 1977 and first installed in that year.

The system contains simple word processing and drives an on-line typesetter with limited functionality (four fonts). There is a letter-quality printer also on-line. The software is comprised of an editor, batch hyphenation and justification, dedicated typesetter driver, and simple dedicated letter-quality printer driver.

The development effort took about 13 man-months, using Intel 8080 development machines and assembly language, with one senior person full-time, another senior person for the design only, and a part-time junior programmer. Minor revisons made to the operating system and file system were performed independently.

4. ANALYSIS

Range of Costs and Schedules

The word processor case studies show that a basic word processor consumes about 3.25 man-years of development time. Adding a typesetter interface, from the other studies, would seem to add about another 1 to 1.5 man-years. These numbers are for pure development time and do not include QA or documentation effort.

The Datic word processor contains about 7200 lines of assembler code, and uses 28K bytes of memory with a 10K-byte data area. Crystalwriter is about 10,000 Pascal statements, about 100K bytes of code, and 30K bytes of data. They each took about 3.25 man-years to develop using senior people. The difference in the code sizes is only partly due to the difference between p-code and assembler. It is mostly due to the differences in the user interfaces. Not only does the Pascal system have a menu-driven command interface but it also allows free-form insertion of characters. The Datic system (and many others such as Wang) require the user to open the screen, do the insertion, then close the screen, at which point the text is reformatted all at once. The decision to use the free-form style of insertion is a significant one because it imposes a considerable performance burden on the reformatting algorithm, but the advantages to the ultimate usability of the system are substantial.

The Quadritek typesetter uses 12K bytes of memory, about 6K lines of Assembler source, for the code which handles the interactive justification (formatting for the typesetter). The Graphic Arts Terminal software uses about 20K bytes total for the entire application. The batch hyphenation and justification program mentioned above ran in 8K bytes of memory.

The differences in level of effort among the typesetting projects is interesting because it points to the fact that batch systems are easier to implement than interactive ones. They require less design, less memory, and less code. Interactive user interfaces are costlier, but they represent one of the more significant advancements in recent years especially in personal computer software.

No particularly extraordinary resources are required for text processing system development. The target hardware should of course be available for direct testing. The importance of a good debugger cannot be overestimated. The operating system overhead must be factored in when computing performance constraints, and it is also important to make sure that the operating system does not intrude upon the application's ability to drive the screen, keyboard and printer to their maximum capability.

Suggestions for a Development Plan

Hardware Considerations

The level of effort required to develop a word or text processor depends [illegible] significant extent on the planned target hardware, not only because of the software effort involved to drive the devices but also for basic design and generality issues. For example, if it is desirable to have the capability of driving many different types of terminals or printers, then general drivers should be designed and developed in order to ease the burden of adding each new device. Some performance will probably have to be sacrificed in order to support the generality - not usually a problem for printers because of their relatively slow speed, but certainly a problem for terminals.

Crystalwriter includes both a general terminal driver and a general printer driver. The development cost premium for these general drivers was approximately one man-month each. As a result of the general drivers, a new terminal and a new printer, both different types from the ones used for development, were brought on-line at a customer site with about one man-week of additional development effort.

The importance of the screen quality to word and text processing cannot be overemphasized. Not only should the physical screen and keyboard be of a modern ergonomic design (tiltable height-adjustable screen, typewriter-style, quick response keyboard with function keys), but the design of the screen characters and the phosphor should be such that they do not cause eyestrain. The IBM PC Monochrome monitor is an example of a good basic screen for text (25 lines x 80 characters, 9 x14 character matrix, relatively slow green phosphor). Most terminals are not suitable for extended word processing use.

From an internal standpoint, the advantages of a memory-mapped screen are considerable. Not only is it easier to map the internal data structures to the screen if it is memory-mapped, but the problems of operating system overhead and terminal overhead disappear. The word/text processor screen should be capable of displaying all attributes required for the final output. If a typesetter or multiple-font printer is the output device then a bit-mapped screen would be a logical choice. For a letter-quality printer, a screen with at least bold, sub- and superscripts, and underlining should be employed. If special characters such as foreign characters or math symbols will be printed, then the screen should be capable of displaying them. One of the real contributions of word processing is the accuracy of the representation of the printed page on the screen. A well-designed screen character font and a good user interface are more important, however, than a full-page display.

Any system with a typesetter should also have a proofing device that can show the typeset line endings because of the slow speed and high cost of using the typesetter itself for proofing.

Overall Design Considerations

Support for proportional characters and for multiple fonts and sizes, if it is desired, should be designed in from the beginning. It is difficult to retrofit later because it is an extra level of complexity to handle variable width characters in terms of both the screen handling and the internal formatting algorithms.

The user interface in general and the command structure in particular are key in the initial design. Some word processors and all formatters associated with text editors require the user to embed formatting commands into the text stream (e.g., .LM 10 sets the left margin to 10). A better approach is to show the effect of formatting commands immediately in the text (left edge of the text changes on the screen and all lines are re-wordwrapped to reflect the new margins), leaving the commands themselves either completely hidden or displayable only on user request. Auxiliary aids can be made available to the user for setting of defaults, editing of format rulers, etc., but for best screen appearance, these should be independent of the text display.

Menu-driven command interfaces can be tedious, although this problem can be overcome by providing an expert level. A form fill-in interface with extensive prompting is another alternative. Whatever the style of the command interface it is important that it be universally applied throughout all commands for consistency. A general command handler will also be easier for development in the long run particularly since different people are likely to be responsible for different commands. A case in point is that in CrystalWriter we neglected to do this, but in the translated version to 8088 assembler a general command handler was added, which made the addition of new commands both easier and faster.

A primary consideration in the design particularly of an advanced text processor is to target the type of end user -- technical people will require mathematical equation editing, outline format, automatic indexing and abstracting, integrated graphics and line drawing capabilities; clerical/secretarial users require a good mail merge facility, integrated reminders, executive calendars, etc.; legal applications require footnotes, references and boilerplate handling; economists and statisticians need superior table handling, equation editing, and graph capability.

The file system needs to have the capability of appending blocks to either end of the file, for scrolling in both directions. Files set up as stacks are useful structures for text. The operating system interface should take into account the performance criteria mentioned above, especially regarding screen and printer control. Printing should take place in background, which is not a problem for most operating systems, but an additional requirement for a word or text processor is a provision for operator intervention to change fonts, feed paper, etc. Some means of communication between the editing and the background printing task must be provided for.

5. CONCLUSIONS

The single most important issue in building a word or text processor is maintaining the performance while providing advanced capabilities and a sophisticated user interface. Given a higher-level language, an efficient compiler, and powerful system resources, it is tempting to assume that performance will not be an issue. Here are two counter examples:

1) Apple Corporation's LISA Professional Computer. This system which grew out of the Xerox Star concept has an advanced user interface employing icons and multiple overlapping windows, and uses a mouse as a pointing device. It has a bit-mapped screen and is based on a Motorola 68000 with 1M of memory. It reputedly took 200 man-years to develop using a higher level language. The word processor component of the system (LisaWrite) has been widely criticized for its poor performance especially in scrolling and character insertion. The software cannot, for example, keep up with a competent typist.

2) The Syntrex word processor. Built using a Unix-derivative and in the C language, the software runs on an 8086 processor with 128K RAM minimum. Syntrex astonished the industry by producing a full-function word processor within 6 months. The initial release, however, had poor performance. Large parts of the system were re-written specifically to improve the performance.

Performance is particularly critical with bit-mapped screens because of the extra overhead of the character-to-bit translation. Algorithms must be designed such that changes to the screen are minimized, or performance will suffer.

More advanced devices with higher resolutions and more capabilities are certain to be available in the future and should be taken into consideration in the design of a text processor. If support of variable-width characters is contemplated, then it should be incorporated into the design from the outset.

The ability of the data base to handle both data and text files is important in any integrated system. The design of the file system and of the supporting operating system and device drivers should take into account the special requirements of text processing.

Careful selection of the target hardware is important both from an ergonomic point of view and also in terms of the development effort required to support it. The quality of the screen and keyboard are essential since it is through them that the user views the system.

The user interface is critical, and should be worked out early in the design stages. It could indeed prove to be an advantage from both an implementation and an end-product quality standpoint to build the rest of an office automation system around that interface. It is particularly important not to sacrifice the power of the user interface nor the overall system performance for advanced capabilities, a common fault of some of the advanced research projects in text processing. A text processor is no better than a typewriter if it has sluggish response.

* CrystalWriter *

Capsule Description

NEWBURYPORT COMPUTER ASSOCIATES

~~27 FAIR STREET~~ 1 Wharf Ln
~~NEWBURYPORT, MASS. 01950~~ Haverhill, Ma 01830
~~(617) 462-9411~~ 617-373 9246

Feb 23,1983

Bill Carlson
Intermetrics Corp.
4733 Bethesda Ave. Suite 415
Bethesda, Maryland 20814

Dear Bill,

This is to confirm our telephone conversation authorizing Intermetrics to copy and distribute all materials copywrited by Newburyport Computer Associates, Inc. which were submitted as part of the Text Processing Systems Technical Area Report. This authorization applies for any purpose connected with the use of that report.

Yours truly,

Ralph O Brown Jr

Ralph O. Brown Jr.
Vice President

* CrystalWriter * CAPSULE

CrystalWriter has been designed to be very easy and intuitive to use. It has its own unique characteristics, however, that may be different from other word processors and editors you are familiar with. The more important of these are outlined here in this capsule description so that you can get up and running on CrystalWriter in a minimum amount of time.

For a detailed explanation of all CrystalWriter features, see the User's Guide. The Getting Started lessons offer a complete tutorial.

1. A NOTE ON THE KEYBOARD

Depending on the type of terminal you are using, certain keys may be local keys only and therefore not operable in CrystalWriter. On the Ampex Dialogue 80, these include ALL BLACK KEYS. Should you hit one by accident, do a <CTRL>-v, the **Verify** function, to rewrite the screen.

Avoid <CTRL>-f, -s, and -p. They can stop output to the terminal. Especially avoid the <BREAK> key: it causes an immediate exit to the monitor, thereby destroying all the current text.

2. COMMAND KEY: <ESC>

<ESC> is the access to all commands. It presents a menu of choices of commands, which are selected by one of the characters (case is significant) highlighted in the menu. Once you select a command, another menu may appear requiring a second single-character choice. All commands display complete directions on the top screen lines as they progress.

<ESC> is also used as a **command exit** and as a **command interrupt** - to interrupt certain longer operations in progress, such as search.

<CTRL>-t **cancels** the current command.

Some commands such as Search and Format Define, present you with a form to fill out. The cursor arrows move among the labeled fields. Normal editing keys can be used within the fields.

3. BLOCK COMMANDS

Certain commands ask you to define the scope of the block of text to be acted upon. The general method is to move the cursor in the forward direction to just past the desired scope of the action and strike <CR>. <CTRL>-t cancels. When a block's scope includes the end-of-document, in some commands the definition is automatically ended for convenience.

Block commands include Format, Indent, Adjust, Attribute, Extract, and Delete.

4. CURSOR MOTION: <ARROWS> and <HOME>

The four **arrow keys** move the cursor in the indicated direction, scrolling the text as necessary when the cursor reaches the top or botttom edge of the screen. The up and down arrows are used for explicit scrolling if **Scroll mode** is enabled with <CTRL>-a. As with all modes in CrystalWriter, striking a mode key again returns you to the original mode. Current modes in effect are displayed on the status (third) line of the screen.

You can flip the cursor travel mode between **LOGical** (text only - the most useful), and **PHYsical** cursor motion (anywhere on screen) with <CTRL>-c.

The <HOME> key is used as a precedent key for **express cursor motion.** It presents a menu of choices. <HOME> followed by one of each of the four arrows jump the cursor to line edges. <HOME> d **jumps to end of document.** <HOME> D **jumps to top of document.**

Other fast jumps - to word, paragraph, screen, etc. are indicated by a single letter for each, and can be combined together. <HOME> must be struck again at the end of the sequence. Upper case letters specify backwards motion, lower case go forwards. All fast cursor motions can be used during block definition in the Block Commands.

5. FIND

Another way to move quickly to a position in the text (most useful for text currently ON the screen), is to use <HOME> f to **Find** forward and <HOME> F to Find backwards. As you type the string of characters you wish to move to, the cursor moves to the next occurrence of the string. This is a literal match. Arrow keys repeat the current Find. Type <HOME> once you get to where you wanted to go, or it will keep finding. <ESC> aborts the Find underway.

6. SEARCH and REPLACE

Although the **Search** command will also move the cursor to the next occurrence of the requested string, Search is most useful as a global Search and Replace, since it has wildcard, case check, and count capabilities. The

Search and Replace strings are both fully editable and are saved until they are changed or the end of the editing session is reached. Searches can be interrupted in progress with the <ESC> key.

The Search command presents a form to be filled out with four fields: Repeat limit (number of times to repeat the Search- blank defaults to infinite), Case Check (No ignores case, Yes is case-sensitive), the Search string, and the Replace string. Move from field to field with up and down arrows. Normal cursor left and right and editing keys can be used within the fields. The wildcards available for the search string, for numbers, alpha, or both, are displayed just above it. <ESC> executes the search from the cursor position forward. The search stops at the start of the matched string if it finds it, otherwise at the end of document.

7. INSERTION

Insert is the default in CrystalWriter: as you type, characters are simply inserted in the text stream, pushing any existing text forward. There is no Insert key. To write over (overstrike) existing characters, you can enter **Overstrike mode** by typing <CTRL>-o.

All lines of text are wrapped on word boundaries except lines which end with '<', the **end-of-paragraph** symbol, inserted with **the <CR> key.** The <CR> key is also used to insert blank lines. A warning beep is given at 5 positions before the right margin. Notice that a special underline character is always present at the very end of the text. This is the **end-of-document** symbol, beyond which no text can be inserted.

Notice that when editing causes a line to overflow, a whole new line is opened to accomodate the overflow word. This is done in order to minimize screen motion; otherwise the rest of the paragraph would be being constantly readjusted on every word. To force a paragraph to be **reformatted,** just do <DOWN-ARROW>'s across the short lines; the text will be wrapped up.

Blocks of text of any size up to complete documents can be inserted at the cursor via the Restore command: see the description of Move Block.

8. DELETION

Character delete is performed with the <DEL> key.
Rubout (back delete) is done with the <PAGE/NEW LINE> key.
Erase word is <CTRL>-e. Words can be erased from any position in the word.
To **Delete a block** use <ESC> d:

Define the scope of the block with cursor forward motion of any type. Blocks to be deleted can begin and end anywhere on a line. The block is highlighted as you go. The cursor can be moved backwards to un-define any portion of the block, and, in fact, the cursor can be moved all the way back past the original start of block, and then moved forward again. End the block definition with <CR>. To cancel the command strike <CTRL>-t. Should you delete a block by accident, you can **Un-delete** it with <ESC> u.

To **wipeout** your entire current text, you can use <ESC> *. The current formats remain. Use with caution! - no second chance is given with Wipeout.

9. UNDERLINE KEY

The **underline** (over the zero) underlines text as on a typewriter. Large blocks of text can be underlined with the Attribute command (<ESC> a).

10. HYPHENATION

Striking the **hyphen key** inserts a hyphen character (minus sign) in the text as you would expect. This type of hyphen remains until you delete it.

Another type of hyphen, the discretionary hyphen, is inserted by the **Hyphen command** (<ESC> -). This command searches the text for lines which are short enough to allow a partial (hyphenated) word to be brought up from the next line. On a user response to the correct position for the hyphen, the partial word is brought up, and the process continues with the remaining lines. The <ESC> key cancels the command.

A discretionary hyphen is removed if, after editing, it is no longer positioned at the end of a line. It is recommended that Hyphenation be performed just before Printing.

11. MOVE or COPY BLOCK

To move a block from one position to another within a document, first **Extract** the block using <ESC> e, then move the cursor to the new position and **Restore** the block with <ESC> r. Both Extract and Restore ask for a name, but you can just type <CR>, and the block will be saved in and restored from the temporary buffer.

To make a copy of a block, follow the same procedure as move block, except that after the Extract, immediately do a Restore in the same place.

Defining the scope of the block in the Extract command is exactly the same procedure as in Delete block - move the cursor forward, the block is highlighted, strike <CR> to end it. The block is removed and saved in the temporary buffer, and the remaining text is reformatted. (Occasionally the screen will be incorrect after a large Extract due to terminal overruns. Execute a **Verify** (<CTRL>-v) to repaint the screen and the attributes.)

In the Restore command, the block will be merged into the text in the current format. The block is available to be restored again any number of times (even into a different document) until another Extract to the temporary buffer is performed, which writes over the existing contents.

12. SAVE and RESTORE BLOCK

To Save a block of text as a permanent document, simply do an **Extract**, giving the block a name. The block becomes a complete document of its own. If it is given a '@'-prefixed, single-character name, the block is saved as a temporary file and will be deleted at the end of the editing session.

Any document, whether created from an Extract or from a Close, can be merged in at the cursor position by using the Restore command and filling in the document name. The restored text takes on the format of its new environment.

13. FORMATTING

With a few exceptions mostly due to the limitations of the specific terminal you are using, all formatting (centering, indentation, etc.) is shown on the screen. Points in the text where a format transition occurs are remembered by the system even after the text has been heavily edited. An 'F' is shown at the extreme left of the first line in the new style.

Formatting changes are specified by commands which allow you to designate the block of lines to which the change is to be applied. There are three of these commands: 1) the **Format** command for explicit change of format, 2) the **Indent** command for indents, and 3) the **Adjust** command for centering, right-adjustment, left-adjustment and justification of lines.

14. FORMATS

A CrystalWriter format consists of a ruler line (containing tab and margin settings) and specifications for line spacing, justification type (center, left, right, or justified), and font. Allowable values for font are 10 (10-

pitch) and 12 (12-pitch). If you use 12-pitch, set the margins at the extreme edges of the screen (columns 1 and 79) to obtain the full page width.

Up to 20 different numbered formats per document can be defined with the **Format Define** command (<ESC> F). A format can be invoked as often as needed using the **Format** command (<ESC> f). The Format Define command presents a form to be filled out. Move from field to field with the arrow keys and use normal editing within the fields. Editing within the ruler line is slightly different, and is described under Tabs below. To change an existing format, place its number in both the 'To:' and the 'From:' fields. A warning message will be displayed once when you redefine an existing format. The source format ('From:') defaults to the current format number at the cursor, but it can be changed. The destination format ('To:') defaults to the next format number available, but it can be changed to any other legal number. Save the new format by striking <ESC>, or abort the Format Define with <CTRL>-t. A new format remains defined until you redefine it. The format is not actually used in the text until a Format command is given with the format's number.

The **Format** command asks for a format number. The scope of the block to be reformatted is then defined in the usual manner for block commands, and lines are formatted as the cursor is moved down. Strike <CR> to end the command.

The number and contents of the format at the cursor position are always shown in the format display in the two lines just above the text. Text can be entered in one format and later changed to a different format, either by changing to a new format number, or by redefining the existing format. Whenever an existing format is redefined, the entire document is scanned for references to that format, and those sections are re-formatted accordingly.

15. TABS and TABLES

Tabs can be one of four types: right, left, centered, or decimal-aligned, and can be mixed at will within a format **ruler.** Tabs can be used informally to position text to an absolute position in the line as in the closing of a letter, or to effect a paragraph indent (as in this document), or more formally to set up an entire table with multiple columns.

Tab stops and margins are set in the **Format Define** command. When the command is entered, the cursor will be in the 'From:' field. An up-arrow will position the cursor in the Ruler line. In the Ruler line, existing tab stops and margins can be removed with the spacebar or with any delete key. The cursor can be moved in the horizontal direction. New tabs or margins are set by typing the character corresponding to the tab type or margin desired on the new position (t=tab, r=right tab, c=centered tab, a=aligned tab, >=left margin, <=right margin). There can be exactly one left and one right margin, and up to 20 tab stops per ruler.

The <TAB> key inserts a tab into the text, moving the cursor over to the next tab position. The text you then type will be aligned according to the type of that tab. In decimal-aligned tabs, inserting a period anywhere within the field re-aligns the text so that the period falls on the tab position. To move the cursor from tab to tab within a table, use the arrows or <HOME> t.

A tab can be deleted with any of the delete keys. Consecutive tabs to the right are all removed at once and the text moved over.

Note that a tab can be thought of as one elastic space, rather than a sequence of regular spaces: a tab is inserted or deleted with one keystroke. Tabbed text is attached to its tab stop and remains so even after it has been edited or reformatted. If characters are deleted, for example, from a tab field, the characters in the next field are <u>not</u> moved closer to the deleted text as they would be if there were intervening spaces. Instead, the characters stay lined up to their respective tab stops and the elastic space (tab) expands to fill the gap. Only insertion and deletion of the tabs themselves, or a format change, alter the position of the tab fields.

To set up a table, define a format containing the tab stops and margins for the table, and insert a few blank lines in the text where the table is to go. Invoke the new format there using the **Format** (<ESC> f) command. Enter the text using the **<TAB> key** to separate the fields.

Existing columns of text can be moved left or right by changing the table to a new format which has new tab positions defined.

You will occasionally see a tilde appear on the screen: this is the symbol for a 'bad tab', a tab which has no tab position to go to, either because there are no more tab positions on the line or because tabbing is illegal on that type of line (centered or right-adjusted lines).

16. INDENTS

Indents can be thought of as temporary margin changes, and are generally used to distinguish certain areas of the text from the rest. A paragraph indent, which indents only the first line of a paragraph (as in this document), is best done with a <TAB> or with spaces.

An indent is defined in terms of one of the tab positions within the current format, but it is independent of the format in that the indent can remain in effect across several format changes.

In the **Indent** command (<ESC> i) the indent position is set by moving the cursor from tab to tab across the ruler using the arrows until the desired position is reached: the letter 'I' for indent moves with the cursor, marking the position of the indent. Strike <ESC> to define a right indent in an analogous manner. <CTRL>-t cancels. <CR> finishes the indent definition and indents the current line. The indent can be continued for more lines with cursor down or any of the forward jumps. End the block definition one line past the extent of the indent with <CR>. To move indented text out to the margin follow the same procedure, setting the indent position on the margin.

New text can be entered indented by setting an indent on a blank line before typing in the text. If text which is already indented is included within the scope of an indented block, the indented text will be indented further by the amount of the defined indent. This relative indenting allows a block of text with multiple indents, such as an outline, to be moved in or out as a unit. If indented text is changed to a new format which has different tab positions, the text will be indented to the tab position in the new format which corresponds in number to the indent's original tab position.

17. CENTERING, RIGHT and LEFT ADJUST, JUSTIFICATION

Lines can be centered, left-adjusted, right-adjusted, or justified (even on both margins on printing), independent of the format in effect. These line adjustments, performed with the **Adjust** command (<ESC> j), override the justification type in the current format for the lines specified. The 'Justify:' field in line 4 always displays the justification state of the cursor line.

Once the type of adjustment is selected from the menu in the Adjust command, the block to be adjusted is defined in the same manner as for indents or formats. One line is automatically adjusted, and more lines can be adjusted by moving the cursor down. The scope of the block is ended at the line following the last one with <CR>. New text can be entered in centered, right-adjusted, etc. by doing an Adjust on a blank line.

18. UNDERLINING, BOLDFACE, SUBSCRIPTS, SUPERSCRIPTS, CASE CHANGES

The **Attribute** command (<ESC> a) permits making blocks of text bold, underlined, double underlined, subscripted, superscripted, lower case, or upper case by the selection of the appropriate letter from the Attribute command menu. Both types of underline can be done continuously or on words only. The 'Attr:' field on the status line always displays the current attribute in effect at the cursor. The block to be attributed is defined with forward cursor motion of any type. Partial words or lines can be attributed. (The attribute of the text is changed as the cursor is moved, but the attribute may not be displayed depending on the limitation of the particular terminal you are using: for example, all attributes are displayed as underline on the Ampex Dialogue 80.) To undo an attribute change, the menu selection letter in the Attribute command is entered in upper case: for example, to erase boldface enter 'B'. The same text can have multiple attributes, but each attribute must be set individually.

The **Mode** command (<ESC> m) sets the text entry mode to one of the attributes. All new text entered will be inserted in the new attribute. If the current text entry mode is attributed, the name of the attribute is shown just to the right of the word 'TEXT' on the status line. To return to normal un-attributed entry, strike <ESC> m n.

19. PRINTING and PAGINATING

The font and line spacing values used for printing come from the formats in the document, although the line spacing can optionally be overriden at print time: this overrides only the main document spacing, however. To change font and line spacing values, use the **Format Define** command and move the cursor to the appropriate field and edit the value. Line spacing can take the values: s=single, d=double, t=triple, h= one-and-one-half, q=one-and-one-quarter. Fonts are either 10-pitch or 12-pitch. Use the Format command to change the format of the text.

Before printing a document, a header-footer can be set up using the **Header** command (<ESC> h). If you want page numbers, or top and bottom page margins different from the default values (one inch), be sure to define a header-footer. If there are no header-footers defined, the document will be printed without page numbers and with one-inch top and bottom margins.

The Header command first asks if you wish to edit the header/footer for all, first, odd, or even pages, then removes the main document temporarily from the screen and displays instead the header-footer requested if one already exists. If not, an empty screen is presented. When you are editing a header-footer, the word 'HDR' followed by an identification of which header it is, replaces the word 'TEXT' on the status line. Enter <CR>'s for each blank line of margin desired at the top and at the bottom of the page. Separate the top (header) from the bottom (footer) with a Soft or Hard Page Break. Page breaks are inserted via the **Page** command - <ESC> g, and are deleted with any delete key. Text can be placed in the header and footer using all normal editing and formatting commands. The Format Define command cannot be executed during header-footer editing, but all defined formats are available.

The desired position of the page number on the page is specified by inserting a Page Number Variable (via the **Page** command) in the header or footer or both. It appears as '#' on the screen and can be deleted with any delete key. It acts like a piece of text, so it can be centered, flushed right, etc.

To return to the main document, use the **Close** command (<ESC> c). When you **Print** (<ESC> p), a series of menus of options for printing and pagination will be presented. Progress from menu to menu with the spacebar. The options include parameters for form length, widow and orphan avoidance, page numbering, and number of pages to print. The left margin adjustment parameter is available to offset the left edge of the print image on the page for any reason including alternating odd-even page offsets for special binding requirements. There are also switches for paginating without printing (Insert-Page-Breaks option on, Print option off), manual pagination (Insert-Page-Breaks option off), saving or not saving the paginated document, and single sheet feeding. Defaults can be taken for all options.

Print prints the document and, unless manual pagination has been selected, inserts Soft Page Break lines at every page boundary, which can later be jumped to using the <HOME> g command. Existing Soft Page Breaks are removed, but Hard Page Breaks remain. Print can be interrupted with the <ESC> key. A message is displayed when Print has completed, and editing can then resume.

20. VARIABLES

To insert variables in a template letter for mass mailings, use the **Variable** command (<ESC> v), which asks you to name the variable, and then inserts it at the cursor position encased in back slashes. To create the individual letters, open the template letter, and using the **jump to variable** (<HOME> v), delete the variable with <DEL>, and type in the appropriate value, repeating for each variable. During Print, the letter will be totally reformatted and repaginated.

21. KEY SAVES

Ten keys can be defined by the user, via the **key save** command (<ESC> k). Enter a number 0-9. Now you are in key definition mode, indicated by the word 'KEY' in the status line. All keystrokes (up to 512) now typed, whether commands or characters, are saved under the number requested (0-9). <CTRL>-q ends the key definition. Keys may call other keys to 5 levels deep. Key definitions cannot be edited.

The key sequence can then be **run** by typing <ESC> followed by the number. <ESC> interrupts the execution underway. Defined keys are saved across editing sessions under a file named 'PROGKEY.DATA' on the user disk.

22. IMPORT/EXPORT

Import of .TEXT files to CrystalWriter format is performed via the **Restore** command using either the S=Source or T=Text options. The Source option adds carriage returns to every line and is designed for importing source code. .TEXT files can be merged into existing documents or brought in by themselves to an empty screen.

Export of CrystalWriter files to .TEXT format is done with the **Print** command- select T for the Output option in the Print (third and final) menu. This will generate a paginated image in Ascii of your document, with header-footers, page numbers and form feeds. To generate a straight image without form feeds and header-footers, make sure all the header-footers have been cleared out from the document before printing, and set the line spacing value to single. The converted image will be *written* to disk as *name*.TEXT. It will not be printed.

23. OPEN, CLOSE, and QUIT

The **Open** command (<ESC> o) retrieves the specified document from disk and loads its text, header-footers and formats. The document is now available for editing. **Close** (<ESC> c) saves the current text under the name you give it, and renames the old version, if any, to name.gback. The **Quit** command (<ESC> q) is the exit from CrystalWriter. Quit, just like Open and Print, gives you a chance to save your text first, if there is any on the screen. To create a brand new file, enter the text onto a clear screen, then execute the Close command when you finish.

If there is insufficient contiguous space remaining in the user disk area to save the entire document, a message will be displayed before the document is saved. Use Extract to save segments of the document to separate named files. This problem can usually be avoided by making sure that there is enough contiguous space on the disk before editing a large document.

CLUSTER II PAPERS

STANDARDS GRAPHICS PACKAGES
FOR COMMAND AND CONTROL

Prepared for: Dr. Tom Probert
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, Virginia 22311

Prepared by: William E. Carlson
Director, Washington Division
Intermetrics, Inc.
4733 Bethesda Avenue
Bethesda, Maryland 20814
and
Stephen Shelley
Senior Computer Scientist
Intermetrics, Inc.
733 Concord Avenue
Cambridge, MA 02138

STANDARD GRAPHICS PACKAGES FOR COMMAND AND CONTROL

1.0 OVERVIEW

This paper discusses standard graphics packages which can facilitate the implementation of command and control system software. While the primary focus is on WWMCCS requirements, the packages discussed would be of great value in most command and control applications.

Two international standards which are achieving widespread acceptance provide a sound foundation on which to build command and control graphics capabilities. The proposed ISO/DIS 7942 Graphical Kernel System (GKS) is a set of basic functions for computer graphics programming that standardizes an application level programming interface to a graphics system. A proposed Ada binding for GKS has already been developed and submitted to ANSI X3. Complementing the GKS standard is the North American Presentation Level Protocol (T500-X3L2.1/82-72), which has been designed to allow the digital communication of graphic information over low-bandwidth channels.

Graphics applications in command and control include report generation and presentation graphics, for decision support, teleconferencing, mapping and map related analyses. More generally, graphics software is becoming a tightly integrated part of the user interface in modern interactive systems such as the XEROX Star and the Apple Lisa. The best way to maintain a consistent user interface and vendor independence across all applications will be to have a set of user interface construction tools based on standard portable graphics building blocks.

The driving performance requirement will be user acceptability in the interactive decision support environment. More specifically, many WWMCCS applications will involve relatively simple displays of graphs, bar charts, pie charts, etc. These applications should be supported on all terminals, including a variety of very low cost terminals with minimal intelligence. The software overhead associated with the generalized device independent interface should not make the system appear sluggish to the users or significantly reduce the number of users that can be supported on a particular mainframe. The ability to transmit these simple graphics over ordinary phone lines using the NAPLPS protocols will be of great value.

A good indicator of the high performance state of the art in graphics technology is the simulation of a day view of an aircraft landing in three dimensions. An approximate simulation can be achieved on a terminal costing $150,000 but a realistic simulation requires a multimillion dollar flight simulator. Such a simulation places requirements on the graphics hardware and software which are unlikely to be found in the command level WWMCCS environment.

The most stressing graphics application identified to date in discussions with the command and control community is the "browsing" problem. Commanders need to flip quickly through documents looking for information that is relevent to a decision they are making. It would often be very difficult for them to say what criteria they are using to identify "interesting" information. As more and more of the information they need becomes most readily available on-line, they will need to be able to page rapidly through on-line documents, skimming as little or as much of the information on each page as they want. The appropriate environment is a bitmapped display containing a mixture of multifont text and graphics. The apparent time to bring up a new page should be one vertical sweep, which is 33 msec. Since it takes at least 100 msec to move 1 Mbit from the disk into a display buffer, some hardware trick is needed to achieve the desired performance. A reasonable solution is a system that has dual display buffers, fills a display buffer in less than a second, prefills with the next page in background mode, and can switch between display buffers during the retrace time.

The remainder of this paper discusses the GKS and NAPLPS standards, the level of effort to implement those standards, and likely WWMCCS functional requirements for graphics packages that extend those standards. The overall conclusion is that it will take a coordinated family of projects involving a total of 25-50 people over a 24 month period to explore quickly and thoroughly the range of graphics alternatives available for the next generation WWMCCS system, and to produce complete efficient and stylistically sound implementations of the required graphics building blocks in Ada.

2.0 DISCUSSION OF FUNCTIONAL REQUIREMENTS

a. Levels of Graphic Output Functionality

A basic level of graphic output capability provides the ability to draw lines and polygons. Augmentations provide for circles, arbitrary curves, area filling, control of color, rotation, scaling, pan and zoom, and text. In the text area, extended capabilities include publication quality fonts, user defined fonts, and the ability to rotate fonts so that text can be displayed at an arbitrary angle across the screen. Advanced graphics capabilities provide for the display and transformation of arbitrary image data, the display of three dimensional objects on a two dimensional plane using appropriate transformations, and the shading of objects to enhance the three dimensional image and to simulate shadows and similar effects.

Displaying moving pictures is more difficult than displaying graphics which are static. The extreme case of dynamic imaging is the aircraft landing simulation, which requires the creation of images that simulate a wide angle camera taking pictures through the front window of an aircraft as it lands. One reason for using dynamic graphics is to portray a dynamic situation. A different reason is in response to user commands. For example, if the user wants to study a particular segment of a graph, the system might "zoom" in on that portion of the graph by simulating a magnifying glass that expands the interesting segment and clips off the rest of the picture. Similarily, the "pan" operation simulates a TV camera scanning across a chart or map that is too big to fit on the screen.

Scaling or rotating a picture is made difficult by the limited resolution of available display monitors. High quality photocomposition systems use more than 1000 points per inch to give the visual impression of continuous lines, whereas a TV monitor provides only 20-30 points per inch and high quality video display monitors provide only 80-120 points. The eye can easily detect the granularity of images at 300 points per inch, so the presentation of diagonal lines and curves on available display monitors is at best a crude approximation. Because of this problem, which is called aliasing, straightforward linear scaling of the points that comprise a line or shape can produce visually unacceptable results. An acceptable graphics system must compute the most appropriate representation of the abstract shape to be presented in terms of the available display resolution and the available colors.

b. Graphic Input Functionality

Graphics input devices include joysticks, tablets, mice, wands and the keyboard cursor controls. The basic input function, which GKS calls the "locator", is to record x-y coordinates. Options are to select a particular position, and to continuously record the path of the pointing device as it moves, which GKS calls "stroke", and to select an object on the screen, which GKS calls "pick". Input readings can be absolute or relative. There can be one or many input devices on the system.

Another possibility is direct input of image data, in video, fax, or digital format. Video data would normally be used in combination with other input techniques.

Productivity in creating on-line graphics can be enhanced greatly with interactive systems that understand what kind of picture the user is trying to create and allow the user to select from a menu of reasonable choices. For example, the user might select from a list of standard graphic formats (say x-y plot, bar chart, and pie chart) and specify parameters rather than drawing the desired shape from scratch. As another example, with appropriate interactive software, curved lines can be drawn by starting with the end points and then pulling a line connecting them into the desired shape.

c. Graphics Hardware Environments

At the bottom end of the hardware spectrum, minimal NAPLPS terminals use standard televisions as the display and provide 256 pixels horizontally by 200 pixels vertically with either four or eight colors. More capable terminals provide 512 by 512 pixels with 16 colors, and high resolution terminals provide 1024 by 1024 pixels with as many as 16 million shades of color. Terminals with 2000 by 2000 pixels are projected to be available soon. Raster displays are rapidly replacing vector displays in all applications, and should be the basis for the next generation WWMCCS graphics capabilities.

Three techniques are used to connect the primary applications program running on the cpu to the graphics output device. Inexpensive personal computers use part of the processor°s memory as the display buffer, and the processor must create a bit pattern in the display buffer that corresponds to the desired display. On the other hand,sophisticated graphics systems have one or more display processors that read instruction sequences placed in memory by the cpu and create the required display. The third choice is to use a telephone line, or perhaps a higher speed communications line, to connect the terminal to the computer. NAPLPS terminals are intended to operate in this mode.

A special Ada run-time system will be needed to control a graphics processor. Also, to the extent that the graphics processor is to be programmed in Ada, another code generator for the Ada compiler will be required. While there is a significant trend towards the use of off-the-shelf microprocessors such as the Motorola 68000 and the Intel 8086 as graphics processors, the highest performance systems still use specially designed microprogrammable processors.

3.0 BASIS FOR IMPLEMENTATION EFFORT SIZING

a. GKS

The GKS standard was originated by the West German Standards Institute in 1978. The draft standard was published in 1982. It consists of multiple upwards compatible levels. The ANSI X3H3 committee is processing the American version of the standard, which is compatible with the proposed ISO standard and adds the binding to specific programming languages. Fortran is dealt with in the current draft, and a proposed Ada binding has been submitted to the committee for consideration.

Harris Corporation, which developed the Ada binding for GKS, estimates that an implementation of the full standard will be about 12,000 lines of code for the device independent portion and about 2,000 lines per device suppported. There will be a high variance in the amount of code per device, depending on how closely a device provides the functions required by GKS.

As a cross check on the Harris estimates, data is available on a GKS implementation produced by AED in Germany. Their system is implemented in FORTRAN for VAX computers running the VMS operating system. They support the Tektronix 4010, 4014, and 4114 terminals. They also support Calcomp plotters and displays from Megatek and Ramtek. Total object code size is 120kbytes, running in a 70kbyte region. As a point of interest, the binary license costs $5,000 per cpu supported, and a single system source license is $30,000. This data is consistent with the Harris estimates given typical expansion factors from Fortran to machine code.

b. Programmer's Hierarchial Interactive Graphics Standard (PHIGS)

PHIGS is a new standard under development to solve various limitations of GKS. Its requirements state that it must be upwards compatible from GKS whenever they supply similar functionality. Major improvements over GKS include support for three-dimensional graphics, improved capabilities for modifying 2-d and 3-d objects, and support for rapid dynamic articulation of objects. PHIGS represents a significant extension of GKS, and when complete will supply all of the functionality of the ACM SIGGRAPH Core standard and more.

Since PHIGS is still in its early stages, it is difficult to estimate the implementation effort. The cost for a complete implementation of the SIGGRAPH Core standard gives an indication. George Washington University has done a complete implementation of the Core standard in Fortran. It includes 40,000 lines of code and 400kbytes of object code. Hence, it is nearly four times larger than the current GKS standard.

We can anticipate that the WWMCCS community will need to extend GKS along the lines of the Core standard, either maintaining compliance with the draft PHIGS standard or going its own way if the PHIGS standard does not evolve quickly enough.

c. NAPLPS

Partial implementations of NAPLPS are commercially available. Code in a terminal to present NAPLPS images ranges from about 40kbytes of object code up to 120kbytes, where the larger implementations are nearly complete. Hence, implementation of NAPLPS code in a terminal involves about the same level of effort as a GKS implementation.

As NAPLPS comes to be a significant factor in the commercial marketplace, chips will be available which implement the standard. Hence, it will be very easy for terminal suppliers to offer terminals that implement the full standard, and DoD will be able to buy terminals at reasonable prices from multiple vendors.

DoD can encourage the convergence of the NAPLPS standard and the early availabilility of NAPLPS terminals by making available a complete model implementation written in Ada.

d. Processing Text on High Resolution Bit Mapped Displays

A variety of systems have been implemented which support high resolution black and white bit mapped displays. Examples are the Apollo computer, the Sun workstation, the Xerox Star, and the Apple Lisa. The experience in the implementation of these systems and the resulting performance suggest that text is a special case that must be dealt with explicitly for any application where it will be an important factor in the use of the terminal.

Standard terminals have a display controller chip which implements a standard font, taking a character code as input and producing the proper image on the screen. In a bit mapped terminal, the characters must be drawn in the bit mapped memory. The advantage is that the system builder has total freedom in selecting the fonts that will be used, allowing the use of publication quality fonts, proportional spacing, and other techniques for making text on the screen look like it came out of a book. The disadvantage is that drawing the characters is a computationally demanding process. Without hardware support, a significant fraction of the cpu cycles can be absorbed just painting characters, and result in performance problems in other parts of the system and/or unacceptably slow response to the users.

Standard functions have evolved for copying characters from a font array onto the screen, and for horizontal and vertical scrolling of text. These same functions turn out to be equally useful for scrolling graphic images and for dividing the screen into independent windows. These functions, called BITBLT and Rasterops, are subsets of the functionality provided by graphics processors. Systems such as the Apollo implement these functions in hardware, providing the desired functionality with no degradation in performance. It should be noted that these functions involve extracting bit strings on arbitrary bit boundaries, so they cannot be implemented efficiently in conventional microprocessors, although a dedicated microprocessor can meet the demands of all but the most performance sensitive applications.

The overall effort to implement BITBLTor Rasterops in software is not large. These functions are identified specifically in this report because they should be available in the graphics library, are likely to be the foundation on top of which a window system is implemented, and consideration should be given to acquiring a high end command and control terminal which provides hardware support for these functions. An alternative would be to acquire terminals with display list processors, which can be used to provide equivalent functionality using different algorithms.

e. GRADS

The GRADS system is a high performance implementation tool for avionics display systems. A prototype was implemented by Intermetrics for NADC. It is an example of the kind of high performance real-time system that is not well supported by general purpose packages such as GKS.

The target hardware environment consists of an AYK-14 computer and a special purpose display processor. A typical application would be the F-18 situation display. The prototype system was used to create a vertical situation display that was updated completely 20 times per second.

The programmer using GRADS defines a display format and a set of expressions that relate the display to data produced by an applications program. Meanwhile, an applications programmer codes the primary application in CMS-2. The GRADS system generates a set of CMS-2 procedures which are used to update the display. These CMS-2 procedures are responsible for updating and managing the display program downloaded into the display processor.

The prototype GRADS system was implemented by three programmers over a 12 month period. The total system consists of 35k lines of structured Fortran. The host for the GRADS development system is a CDC-6600. The programmers working on the project were very experienced in the implementation of graphics systems, and included one of the key people in the development of the ACM SIGGRAPH Core standard.

f. C-Compiler for Graphics Processor

Graphics processors are often horizontal microcode machines with complex and unusual instruction sets. Implementing a standard graphics package in the assembly language of such a machine is a painful but finite project. DoD can reasonably expect the terminal vendor to supply the standard functions. DoD will probably have to support the integration of graphics peripheral processors with the run-time system on the host.

If the performance requirements of the applications require a significant amount of custom code to be written to run in the graphics processor, then attention must be given to the development environment used to develop that code. GRADS was one approach. An alternative is to write a compiler for a general purpose language, targeted for the graphics processor. For example, a C-language compiler has been built that produces code for one vendor's 56-bit wide horizontally microprogrammed graphics processor. One could do a similar Ada code generator given a sufficiently pressing requirement. The results with the C-compiler have been impressive.

For example, a molecular modeling system that creates spheres, does shading and does hidden surface removal was implemented in 3-4 pages of C-language code. Performance with the code written in "C" is adequate; performance with a layered implementation built on top of GKS would be unacceptable.

g. Presentation Graphics

A common use of graphics is for management presentations. Such applications should not require custom programming. Instead, a variety of interactive systems have been implemented which ask the user the form of the desired presentation aids, the source and format of the data, and then produce the required graphs and charts. Quite sophisticated systems have been implemented by a team of 3-4 people over 6-12 months, given a base equivalent to GKS. This level of effort includes the user documentation and testing to commercial product standards. In fact, the primary effort is the design of the user interface and writing the manual, since so little code must be written on top of GKS to achieve the required functionality.

4.0 ANALYSIS

This section identifies specific projects which should be undertaken, with suggested levels of efforts. Each project would produce a result in twelve months. For projects producing a final production quality product, an additional six months would be needed for refinement and tuning. The next step in projects which are exploring alternatives should be decided at the end of the first twelve month development period.

a. GKS (Production Product

GKS should be implemented in Ada. Since typical industry productivity is 2,000 to 3,000 lines per man year, a 5 person team should be able to do the job in a year. The team should be very experienced, and they should create model code rather than doing the job as quickly as possible. Also, device drivers should be written for a variety of different terminal types, including a personal computer of the IBM PC class, a Tektronix 4114 compatible terminal, a high performance workstation, a NAPLPS terminal, and a high resolution color graphics display such as the Ramtek or the Lexidata. In addition to the basic development team, about one person will be needed for each class of terminal to do an adequate job of defining the similarities and differences among terminals in the class and reflecting that parameterization in the Ada model. Hence, the total level of effort will approach ten people on this project.

b. PHIGS (Experimental System)

At least one person should track the PHIGS development. Even better, a team of 3-4 people should start to implement PHIGS and participate in the standards activity in order to help the standard to converge rapidly so it is ready to meet the WWMCCS needs.

c. NAPLPS (Production Product)

A five person team should easily be able to implement a model implementation NAPLPS terminal software in a year. They should also be able to deal with the various classes of terminals in the model implementation, since the NAPLPS standard deals explicitly with all the conversions among terminals.

d. High Performance Graphics (Experimental System)

One or two teams of 3-5 people should work to explore the limitations of GKS and propose solutions for those command and control applications for which GKS will have too much overhead. The GRADS system provides one direction which should be explored.

e. Mixed Text and Graphics Displays With Windows (Experimental System)

Either one or two teams of 3-5 people should work to define the primitives that will be used to implement user interfaces on high resolution bit map displays. The Xerox Star and the Apple Lisa are two examples of the desired kind of user interface. The WWMCCS community should have a standard set of packages for implementing such an interface. With such a tool kit, standard user interface conventions can be maintained across applications and at the same time the WWMCCS system can be vendor independent and able to benefit from new cost saving and performance enhancing technologies from the terminal suppliers.

This required effort is in addition to the effort required to implement a screen oriented text editor, formatter, and other text manipulation tools. The text processing requirements are discussed in a separate technical area report.

f. Maps (Experimental System)

Maps are central to command and control in a variety of contexts. The technology for two and three dimensional mapping is very advanced. An effort should be initiated to implement in Ada the key state of the art algorithms for representing, creating, inputting, transforming, analyzing, displaying, and printing maps. The initial version of routines should be built on top of GKS, but a likely result is that additional Ada packages optimized for mapping will have to be added to the library. This is potentially a very large effort. A team of 5-10 people would be able to do the basic design work and implement a few of the most important packages. The effort can then be expanded after the full GKS implementation is available and after the strengths and limitations of the Ada implementation of GKS have been explored.

g. Presentation Graphics (Production Product)

A team of 3-6 people should be tasked to implement an executive presentation system for creating bar charts, pie charts, simple graphs, and other reporting formats used in the WWMCCS community. There should be a heavy emphasis on requirements analysis, with a thorough search to find all the different presentations formats which must be supported. The system should support transparencies, color plotters, 35mm slide cameras, and overhead projectors as output devices. This system will be one of the primary benchmarks for testing the performance of the GKS and NAPLPS implementations.

h. Teleconferencing (Experimental System)

1-2 people should begin to plan the use of NAPLPS and the presentation graphics software in teleconferencing applications. This will involve the definition and demonstrations of a variety of scenarios, and provide the foundation for the early implementation of required WWMCCS teleconferencing capabiltities.

i. Integrated User Interfaces (Experimental System)

Bridging the gap between the system command language, the graphics and text handling tools, and the applications developer are the family of tools for assembling the user interfaces of applications systems. Appendix A discusses the rationale for separating the user integration tools out as a separate family of modules. A 3-5 person project is recommended to develop a prototype system in this area.

5.0 CONCLUSIONS

Graphics will play a key role in the future WWMCCS system. "Glass teletypes", by which we mean 80 character by 24 line CRTs that are used as if they were teletype terminals printing on paper at 10cps or 30 cps, are already obsolete. The future system should incorporate a variety of graphic terminals, ranging from NAPLPS terminals for the distribution of decision support information over low bandwidth communications lines to high resolution color bit mapped terminals for analyzing geographic data, simulating operations, and other applications.

The range of hardware alternatives, and the need for a family of software packages covering a very broad range of applications, means that the WIS office must quickly explore a very large number of alternatives. Failure to do so could lead to a large number of incompatible user interfaces in the future WWMCCS environment, and also seriously compromise vendor independence.

The proposed development program involves the immediate implementation of existing national and international graphics standards in Ada, and the simultaneous exploration of a few key applications as performance benchmarks for the standard packages. It is likely that the WWMCCS community will need generic packages for the efficient manipulation of documents containing mixed text and graphics, and for the processing of maps. These special purpose WWMCCS packages may also suggest specific functional requirements on some or all of the terminal hardware that will be procured. The recommended development efforts and experiments are all high priority critical path items for the WWMCCS program. Ideally, as many as 50 people organized into about 10 small teams will be started to work in these areas immediately.

Appendix A

A USER INTERFACE MANAGEMENT SYSTEM (UIMS) FOR WWMCCS

Current computer graphics research has indicated that several benefits will arise from management of a user's dialogue with an application system as a module separate from the application package and the graphics output package. Benefits of using a User Interface Management System (UIMS) include

1. better utilization of hardware interaction capability (less susceptible to Least Common Denominator Syndrome),

2. reduced application development costs due to increased use of man/machine interface prototyping and sharing,

3. increased reliability through use of a common well-debugged package, and

4. user interface compatibility across applications.

UIMS(s) of various types and styles have been built are being used in both research and commercial settings. The UIMS proposed in this paper, the UIMS/DF (User Interface Management System / Data Flow), is an extension on these systems in that its architecture provides for both external (UIMS invokes the application) and internal (application invokes the UIMS) control in a multi-thread environment. Multi-process "window" workstations are supported in a natural manner.

The UIMS/DF software views the user/application interface as streams of data flowing in either direction (application -> user, or "output", user -> application, or commands/input). The User Interface Programmer can define and control this flow by connecting, using "Pipes", instances of "Modules" which provide data transformations, data routing and selection, peripheral input (keyboard, tablet, mouse, etc.), terminal display (A/N text, graphics), and application software interfaces. The Modules are selected from UIMS/DF libraries of "standard" Modules, or may be "custom" programmed using Ada. Low level support for the graphic display and input peripherals is provided by GKS functions, or "escapes" when required. The Modules "fire" or execute whenever data is available on their input Pipes.

Several basic benefits of using UIMS tools have been identified in the literature (and mentioned above). Inherent in the recognition of these benefits are the observations that human input interaction techniques are not nearly as portable (from a user-friendly point of view) across display terminal technologies as image display techniques. The logical input devices supported by GKS are well adapted for describing the types of interaction data required, but do not directly support the utilization of good human-machine interaction techniques. For instance, a "Choice" input for a given application may be more appropriately selected via function keys on 1 type of display, while use of a mouse selected "Pick" to implement a "Choice" from a displayed menu would be more natural and comfortable on a different display or in a different context. Other observations include:

1. Interaction technology still proceeds in a trial and error manner as application/user appropriate mechanisms are determined; a process better controlled if the interaction system can be maintained separate from the rest of the application system for easier prototyping and maintainance.

2. Applications built utilizing internal control of input (application invokes input procedures, such as in standard graphics packages) tend to make "backing out" of a command or function relatively difficult.

3. Input techniques buried in the "guts" of an application tend to be difficult to share among other applications, as the various pieces are scattered throughout the software.

In addition to the above general benefits of using a UIMS, the data flow approach adopted by the UIMS/DF provides the following additional advantages. UIMS/DF-built systems are by definition modular with regard to both the application softwareand internally with regard to other Modules. Side-effects are effectively eliminated as inter- and intra-UIMS communication is only via Pipes, there is no shared memory. The "building block" approach in developing user interfaces promotes rapid, easy prototyping of interfaces and convenient mechanisms for monitoring and tracing the flow of information through the system. The functionality of the UIMS/DF is extensible in a standard way -- build new Modules. Display terminal resource handling for multi-process window support is provided by a Terminal Manager built from UIMS/DF Modules and Pipes. The Terminal Manager controls the input/output data by acting as an initial "filter" on all input peripherals and the final output filter for display output.

UIMS(s) are rapidly being accepted as a required component and tool for developing interactive, application systems. User Interface Management Systems of varying degrees of sophistication are being developed and used in several research settings including George Washington University, the University of Toronto, and Arizona State University. Use of UIMS technology during system development is being pursued at TRW and Boeing Computer Services Company. The use of data flow technology is reflected in the UNIX pipe mechanism and more specifically in signal processing applications software. The Evans and Sutherland PS-300 graphics system completely relies on a data flow model for the programming of local peripheral handling and graphics transformations.

Command Language Design

Thomas Kaczmarek
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
14 September 1983

Overview of Command Language Component

This report outlines a package of components to be written in Ada that will provide a command language interface for command and control systems. This package of components should be useful in building interfaces to a system executive or to individual applications programs.

Command and control languages have matured in recent years. Significant trends include:

1. control abstractions that have been found useful in programming languages (iteration, conditional execution, etc) have been assimilated into command languages,

2. graphical and two-dimensional languages (icons, menus and forms) have emerged as alternatives to conventional character string based languages,

3. the command language interpreter has become a service that is accessible from application programs,

4. the command language is considered to be the language of the dialog between the computer and the human--including both commands and responses.

These trends should be included in the product.

Brief description of product

The functionality of the package is divided into six major areas.

1. Form support

2. Menu support

3. Icon and advanced graphics support

4. Command-string support

5. Error message service

6. Keyboard input services

Form support

The paradigm of form filling has gained wide acceptance, especially in applications with very heavy data entry requirements. The value of this interactive technique has also been demonstrated in low bandwidth data collection for control applications. In this kind of application, the major advantage of the approach is that it aids the infrequent or unfamiliar user by prompting for parameter values. It is important to also emphasize the utility of a forms package for the presentation of information as well as the collection of it. The forms package should become the principal vehicle for presenting the computer's side of the interactive dialog.

Requirements for forms support include:

- a form description language that specifies things such as the placement of text in a two-dimensional screen, graphic attributes of the text, protection information for the fields of the form, etc.
- a form compiler or interpreter to directly or indirectly produce actual displays at the user's terminal, and
- a form interaction editor that allows the use to navigate around a form with cursor controls and to enter and correct data entries.

The forms package should exhibit a number of important features. It should contain an integral multi-level help system. The help system can be built using the forms package itself as the methodology for the presentation of help. Forms should be parameterized to allow an efficient technique for the application programmer to pass information to forms and to receive updated results. Forms should be scrollable and support multiple virtual screens. Finally the forms package should be written to be device independent and to make use of information stored in a terminal-capabilities file to convert device independent graphic commands into device specific ones.

Menu support

Menus are another interactive technique that have gained great popularity. They provide the user with a clear depiction of alternatives.

The major requirements for menu support are a menu description language and run-time support. The menu description language should support the description of structured dialogs. That is, the description of the menu should include a specification of what the system should do next for any menu selection. The next action may be the presentation of another menu or the execution of an arbitrary routine. The run-time support should allow the user to back-up through a sequence of menu selections or to look-ahead.

The entire menu package could be embedded into the forms package. This would allow them to share help facilities, device independent properties, and graphic capabilities.

Icon and advanced graphics support

Modern bit-map and color capabilities can be of valuable assistance in the presentation of information. High powered workstations should be part of the future planning for command and control systems. Support must exist for constructing bit-map displays of forms, menus, icons, and text in various fonts. This portion of the product must interact with the forms and menu packages making use of the device independent nature of those components.

Command-string support

Although there have been numerous advances to command languages that have led to alternate command specification paradigms, there still is a place for the more conventional command-string style of command specification. The major advantages of command strings are efficient entry of commands for experienced users and the ability to execute stored command procedures.

Command procedures should be parameterized and contain conventional control abstractions. The syntax of the command language should be compatible with the syntax of the dominant development language, Ada. The command procedures should also include support for exception handling. Functional combination has proven to be valuable in certain situations. Pipes and filters from the Unix environment demonstrate the acceptance of this facility. Functional combination should be provided using a more conventional notation--the syntax of Ada function composition.

Since application programmers may choose to implement a command-string interface, language description and parsing facilities could be made available on a system wide basis. This requires a language definition capability and run-time support.

Error message system

In order to encourage uniformity in the method of presenting errors throughout the system, a error message facility should be constructed. This facility should not only provide a storehouse for messages and routines for displaying them in a standard format, but also, implement standard procedures for reacting to errors and continuing from them.

Keyboard input services

There are a number of techniques related to keyboard entry that should be provided in the product. These are spelling correction, name completion, and input buffer editing.

A general facility to correct spelling errors should be available. Since most command situations involve a very limited vocabulary, this facility can be cheaply added to the system and greatly improve the user interface to command and control applications.

A name completion facility is also quite useful in the design of user interfaces. With such a facility, the user strikes a name-completion key after typing the first several characters of the vocabulary item. If the characters typed to that point are a sufficient prefix for a unique vocabulary item, the system will complete the item. The limited vocabularies associated with command and control make this technique relatively inexpensive and efficient.

Efficient use of these techniques requires that the system be able to limit the possible vocabulary alternatives at any point in the parsing of the input string. This has implications on the parsing routine used. The name completion and spelling correction facilities should be available in conjunction with the forms package to allow their use during interaction with forms.

The product should supply a character string editor that uniformly handles all keyboard input. This editor needs to have only limited capability because it will be assumed to be editing a single line of text. Capabilities should include, deleting the previous character, deleting the previous word, and deleting the whole line. More advanced features like cursor control and insertion could be added. This component should be device independent and use the terminal-capabilities file to derive device specific commands to perform editing.

Conclusion

The facilities described in this report are the result of examining the capabilities of a number of systems. These include. DEC's FMS forms package. The UNIX operating system. VisiOn, Apple Lisa, various XEROX software products (Star, Smalltalk, Interlisp-D) and the TOPS-20 operating system.

The functional requirements for the forms package is heavily influenced by the FMS package developed by Digital Equipment Corporation. Research efforts at USC/Information Science Institute and Carnegie Mellon University have also been influential. These efforts all indicate that the development of such a tool is a reasonable goal to pursue in the time frame recommended for this product.

Menus have occupied a significant position in several XEROX software products. XEROX's use of menus has had tremendous impact on subsequent designs. The menu package requirements have been derived from studying the FMS package, the XEROX products, and several research efforts. The Promise and ZOG systems and work on a menu package done at General Motors Research have provided insight into this component.

Command-string support requirements were derived from studying Unix and a number of other operating system command and control languages. The main lesson learned with respect to command-string languages during the past several years is the importance of control structures in command procedures.

XEROX has been the dominate force in building advanced graphic systems for use with bit-map displays. Smalltalk, Interlisp-D, Star, and Mesa have all relied heavily on this technology and exploited it slightly different ways. The Apple Lisa and Visi On from Visicorp have been greatly influenced by XEROX's pioneering work. All of these systems have influenced the functional requirements for this aspect of the product.

The keyboard entry services have been derived mainly from the TOPS-20 operating system and Interlisp but some of the functionality is certainly available in other products and has been studied in research efforts.

Level of effort

In sizing the effort to produce the command language component of the product, the forms and menu components have been treated together. This integration produces a man-power savings and should produce a superior result. Taken together, the menu and forms efforts should represent about 10 man-years of effort. Much of the design in this area can be gathered from studying existing systems.

The advanced graphics support represents the greatest technical challenge. Only a very few such systems have been built yet. The effort to provide this service is in the 10 man-year range. This is predicated on the package being integrated with a device independent form/menu package. Although there aren't many such systems in existence, a few of them are well documented and this should help greatly in the design of such a system. The Visi On package, for example, is well defined so that external software vendors can integrate arbitrary software products into the Visi On environment.

The command-string support is also about a 10 man-year. This is an area where there is abundant experience waiting to be tapped.

The error message and error handling system requires a relatively minor level of effort. It is roughly a 2 to 3 man-year project.

The keyboard interface component of the system is also a relatively minor task and all three aspects should collectively take about 3 to 4 man-years.

Discussion of functional requirements

This section will present some more details on the functional requirements of the various components of the command and control services. It will suggest some possible alternatives and attempt to point out possible trouble areas.

There are two main concerns in specifying the requirements for this product. One is the need to use the components to produce user interfaces that are friendly and helpful. The second is to make access to the facilities simpler for the applications programmer. The concerns expressed in the following generally fall into one of these two categories.

An important consideration in the design of these components is the trade off between consistency across applications and flexibility of the components. Increased generality and flexibility in the packages will result in application programs that have very unique and distinctive interfaces. Restricting the generality will result in more uniform interfaces throughout all applications. Too much restriction, however, will result in difficulties for the applications programmers.

A help facility must be part of the design of all the major components of the product. The forms package should include an integral help package. This allows the user to press a help button on the keyboard at any point in the interaction with the form. The system response to the help should be in several levels starting with a very brief, one-line description of what is required. If the user is not satisfied, a second press of the help key should give a more complete description of what is expected. The one-line help should appear in some designated prompt area on the user's screen. The more complete help can be implemented by specifying a help-form. This allows the forms package to be its own help system.

The forms package should also allow parameterization of forms. These parameters would be used to pass data between an application and a form. Many existing forms packages do not support this feature and the result is a significant burden on the applications programmer. For output, in existing systems, the programmer must first display the form without any data. convert the data to text, and then direct the forms package to display the text in the appropriate fields of the form. For input, the programmer must first display the blank form, ask the forms package for input, receive a character string, determine which field it came from, and convert it from text to its internal representation. A possible technical advance could be achieved by making the parameters of forms typed using the Ada data typing mechanism.

Scrolling is an important feature to include in the forms package. Frequently all the data to be presented does not conveniently fit on a single screen. The applications programmer must be given some support to make scrolling a relatively easy task.

Since it is probably infeasible to require all users of the system to subscribe to the use of a single type of terminal/workstation, the forms package should be constructed to be device independent. A terminal-capabilities file should exist to allow the package to convert generic graphic commands into device specific commands.

Another useful feature that should be considered for inclusion in the forms package is support for multiple virtual screens. This is an invaluable aid to the user who needs to manually coordinate information.

In the interest of making menus easy to use there are two pitfalls to avoid:

1. deeply nested control trees are difficult to maneuver in, and
2. experienced users frequently are frustrated by the need to go through many menus when a simple command might suffice.

Nonetheless, menus are a valuable tool for structuring command and control of applications. The design of the menu package can be used to minimize the abuse of menus. A menu system with all the power of Promise or ZOG will probably result in menu interfaces that overload the end-users capability to handle them. Restriction of the dialog description capabilities of the menu system can be used to insure the generation of realistic dialogs.

There are two basic techniques for menu interaction. The first could be called the numbered-option approach. Each item of the menu is identified by some unique character, usually a number, that is entered by the user to make a selection. The second approach could be termed the positional approach. Here the user positions a cursor at the menu item and strikes some key or button to indicate the selection. This second approach typically involves using a pointing device on advanced terminal/workstations, and cursor controls on more conventional terminals. Either approach is acceptable. The first approach is somewhat simpler to implement and may be easier to interact with. Thought should be given to designing the system with both and placing the choice of which technique to use in the device dependent part of the menu package. For dumb terminals, the numbered-option approach would be used. For advanced work stations, the positional approach would be employed.

The advanced graphics component called for in this report is similar to window packages that are now emerging in several products. Many people are predicting that this type of interface will dominate all future command and control interaction. Although it represents the greatest challenge and risk, the potential rewards are clear. This style of interaction greatly enhances the learnability of systems and the productivity of users. Because it is a substantial effort and risk, the development plan might have to call for its introduction into the system shortly after the end of the near-term project goal.

Important features of the advanced graphics component are multiple (and possibly overlapping) windows, several predefined fonts, bit-map imaging capabilities, integrated menus (including pop-up menus), and functionality to support shaping, positioning, and other window management operations. Provision for the use of color is important for future growth of the product.

Case studies

Rather than give detailed studies of all the systems that influenced this report, in depth reports will be given for only two major systems. Less specific treatment will be given for systems that are either more conventional or have less influence on the product. Unfortunately, hard data about lines of code, project team size, and level of effort is not readily available for some commercial products. In these cases, level of effort estimates are based on personal experience with similar systems.

FMS

The FMS package distributed by Digital Equipment Corporation is a fairly complete forms package which cleanly interfaces to any of the programming languages supported in the VAX/VMS environment and the PDP-11/RSX environment. It consists of a description language, an editor, a library utility, and run-time support.

The form description language, which is part of the newest release of this package, can be used to construct forms by specifying layout and graphic properties. In the past, the forms editor had to be used in conjunction with a VT100 series terminal to describe a form. The editor is still available and is a useful tool for users with a VT100. The library system provides a method for organizing and collecting forms. It also provides utility functions such as copying, deleting, etc. The run-time support is achieved by linking the application program to a small set of system supplied functions. These functions request the display of a form, direct data to be displayed in a particular field of a form, collect data from a form, etc.

The FMS system allows control of the graphics capabilities of the VT100 (and VT52) series of terminals (e.g., inverse video, high-lighting, and blinking) and with the recent upgrade it can now support the advanced graphics features of the VT100 family (e.g., double width and double height characters). The package has an integral help facility which uses the forms package itself to implement multiple-level help. FMS allows the applications programmer to define protection for fields of a form including specifying a "supervisory mode". The package also handles the definition of default values. The run-time support includes cursor control, automatic field advance, both insertion and replacement mode of keyboard entry and the ability to toggle between them. Scrolling is also supported by the FMS package as is some limited restriction of possible input values (numeric, alphabetic, or alphanumeric). The recent enhancements include attaching routines to fields of a form that process keyboard inputs directly. In the past the applications program was responsible for explicitly fielding all keyboard input.

FMS has been a product of DEC for several years and it was significantly upgraded with a new release this summer. The system is available for two DEC products, the PDP-11 under the RSX operating system and VAX under the VMS operating system.

The size of the effort and development team characteristics are unknown. Experience with similar systems indicate that it was probably a 10 man-year effort to develop this system. Performance enhancements and recent extensions to the system make actual level of effort estimates very difficult to construct.

Visi On

Visi On is one of two recently introduced products that bring advanced graphical interfaces and workstations into the office. It follows the introduction of similar interfaces in several XEROX products. The Visi On package is an attempt to supply just the sort of tool that this study is focused on. The idea behind Visi On is to supply a standard set of interaction techniques that applications will be built upon. Visi On was developed under the direction of William Coleman at Visicorp.

Visi On represents an extensive package of interface techniques. It include forms, menus, windows, and even some aids for memory management for personal computers. It includes a help system and tools for building interfaces to applications. Standardized window manipulation routines are supplied to the end user in such a way that the applications programmer need not worry about them. Thus the user can move a window from one place to another with no intervention by the application program. A major design goal of Visi On is to supply a consistent interface to a number of applications.

Visi On graphics capabilities include multiple overlapping windows. Support exists for controlling graphic properties of windows. An important feature of the product is a what you see is what you get style of editor for interacting with the display.

This system should be a rich source of design ideas both in terms of the interface it supports and the tools it supplies to help define a particular interface.

Visi On was developed over a two and a half year time frame. The first year was spent defining what the standard interface should look like, discovering what basic capabilities to put into the system, and defining the system architecture. A prototyping team of five people built the first version over a three month period. The development phase for the system lasted about a year and a half and represents about 20 man-years of effort. The actual figure is somewhat difficult to attain because of time spent on the development of specific applications, memory management techniques, etc. Most of the system was written in C and it resides in about 128K bytes of memory.

Other systems

The two previous case studies dealt with the more complex elements of the product. FMS serves as an integrated menu/forms system and Visi On is all of that plus advanced graphics. The functional requirements for command-string support, error message facilities, and keyboard services are a synthesis of ideas from several systems. A full case study of each is inappropriate since in most cases only some features of a large system have been used. Fortunately, the technology is much better understood in these more conventional areas and details of the case studies are probably less interesting anyway.

Systems such as Unix and TOPS-20 can be used to provide guidelines about the more conventional aspects of the command language component. The Unix system represents many man years of tuning and refinement of a particular design. The Unix Shell currently represents almost thirteen thousand lines of C code and resides in about eighty thousand bytes of a VAX. The total effort put into the Shell is a somewhat meaningless statistic in this context. Based on the size of the Unix Shell and experience with other systems such as TOPS-20, the man-power requirement to produce the command-string support, the error message handler, and the keyboard input services, is probably around 15 man-years.

Analysis

All of the estimates given for level of effort have included time for design and specification of the system as well as development, debugging, and testing. They do not include any extensive amount of performance tuning or anything beyond initial development.

As was mentioned earlier, the most challenging aspect of the project will be the advanced graphics interface support. This is one area where performance tuning may be critical. There is significant overhead involved in operations involving bit-map displays, especially in the use of proportionately spaced fonts. The quality of the development team in this area is important. The team must have sensitivity to user interface issues as well as some experience building a similar product. Nearly everyone who has built such an interface has seen the wisdom of hiring someone with experience from XEROX.

The development of the form/menu/advanced graphics support systems must be managed in an integrated way. Coordination between these packages is critical. The design of all three must be carried out in synchrony. To the extent that the keyboard entry package and error message components must be integrated with these other packages, their development must also be synchronized. It may be a requirement that all of these components be developed within a single organization.

Conclusions

It is important that the product pay close attention to the recent trends in interface design and support them in a complete command and control language design. Control languages are no longer simple one-dimensional strings with only one-way communication. They now include two-dimensional character-based and graphics-based interaction techniques. This report pays attention to the three dominate paradigms for command and control: conventional command strings, forms, and menus. In addition, it attempts to include more advanced and highly graphical realizations of forms and menus.

It is important that the product pay attention to the needs of the end user as well as the application programmer. This means that the product must include provision for design tools for interfaces. The functionality of the components must be clearly defined and provide enough functionality to take much of the burden of interface design off the shoulders of the applications programmer. The product must also provide for the needs of the end user. Help facilities must be an integral part of the design. Tools must be designed to encourage good interface design. They must provide assistance in areas such as remembering names and spelling. Known techniques for enhancing the user interface (multiple windows, scrolling, etc.) must be supported in the product.

The product will be divided into six major areas: forms, menus, advanced graphics, command strings, error message services, and keyboard entry assistance. All of these areas require close coordination. The greatest technical challenge lies in the area of the advanced graphic interface. Special talent may have to be recruited to insure a successful completion of this component. The total level of effort needed to design, develop, and test the package is roughly 40 man-years.

DISTRIBUTED SOFTWARE ENGINEERING CONTROL PROCESS

TASK 2 ANALYSIS AND FUNCTIONAL DESCRIPTION

SOFTWARE ENGINEERING ANALYSIS

CONTRACT NO. MDA 903-83-C-0202

to
WIS JPM
Technology Directorate
Washington D.C. 20330

29 JULY 1983

from
GTE Network Systems R&D
2500 West Utopia Road
Phoenix, Arizona 85027

PART I

ADA COMMAND LANGUAGE CONCEPTS

DRAFT

(To be included in SEA)

August 5, 1983 - 08:51:46
AR/GSM

DISCLAIMER

This is a draft copy of one section of the Software Engineering Analysis concept paper. It depends, to some extent, on the context of the entire SEA paper. As a stand-alone document, there may be some confusion that is clarified by the SEA itself. The primary purpose of this concept paper is for preliminary feedback on the basic concepts for an Ada command language

CONTENTS

PART I -- Ada Command Language Concepts

Chapter 1

ADA COMMAND LANGUAGE

This section of the software engineering analysis provides the background for selection of the command language used for the DCP. The selection consists of two major parts:

- An analysis of the requirements for a command language (CL) and command language interpreter (CLI).

- A set of models to illustrate the concepts behind the components of the DCP user interface, which are:

  - Ada as a command language.

  - An Ada command language interpreter.

  - A user session illustrating the use of Ada and the role of the interpreter.

  - A menu system model, which allows a high-level interface with the Ada command language and the DCP system.

The Ada command language (ACL) and Ada command language interpreter (ACLI) define the use of Ada as a command language and the concepts behind a command language interpreter. Tables 1 and 2 list the objectives of the sections reviewing the requirements and models for the DCP user interface, before beginning the detailed review of the points.

TABLE 1

Objectives of the Command Language Models

ACL Model	CL Model
1) Identifies the Ada language features that can be used in a command language.	1) Defines features that a language must have to be used as a CL.
2) Defines how the Ada language features will be used.	2) Defines how language features may be used.
3) Defines a conceptual model of the use of Ada language features by the user.	3) Defines the concepts of how a language is utilized by a user.

TABLE 2

Objectives of the Command Language Interpreter Models

ACLI Model	CLI Model
1) Defines how the DCP ACLI will use Ada to interface with the VAX and other systems.	1) Defines how a CLI uses the CL to interface with the system.
2) Defines features that the ACLI will offer that are not part of Ada.	2) Defines functions that belong to the CLI and are outside of the CL.

1.1 REQUIREMENTS FOR A COMMAND LANGUAGE

Requirements for a command language(CL) involve both the language definition and the design of the command language interpreter(CLI). This section presents:

- A review of the requirements for a language that is used as a command language
- General requirements for a command language interpreter
- An analysis of the use of Ada as a command language
- The use of Ada by a command language interpreter

The requirements section results in a list of requirements for the command language and interpreter that are later used in the description of the Ada command language, command language interpreter, session and menu system models.

1.1.1 General requirements for a Command Language

The general requirements for a command language are listed in table 3. This set of requirements results in a language that supports a user interface with an operating system and application programs. Each major requirement is detailed below.

Uniform Interface

A command language (CL) is the interface between a user and the underlying operating system. The user of a system manipulates objects, schedules tasks and collects information using a command language in much the same fashion that a program performs the same functions. The CL should be simple to use and not require significant new learning on the part of the user. The CL should also provide a uniform interface to the user such that the user's perception of the system is consistent, whether the user is creating program source code or communicating with the system via the command language. Uniformity reinforces the user's view of a system, as constrasted with presenting diverse languages to the user.

Ease of Use

A command language should make it easy for the user to instruct the system. The language definition should allow for

TABLE 3

General Requirements for a Command Language

- Present a Uniform User Interface
- Be Easy to Use
- Support Problem Solving
- Support an Object-oriented view of the system
- Support multiple versions of commands and selection of such versions
- Be rehostable, i.e. avoid computer-system dependencies

an easy, abbreviated interface, although much of the support provided to the user is an attribute of the command language interpreter and not of the language itself. As an example, some command language interpreters will accept an unambiguous truncation of an identifier and substitute the entire identifier. The language definition does not support the truncation, which is solely a function of the command language interpreter. The CL user support should reduce the amount of information that the user must manually enter to the system by providing:

- Default parameters
- Shorthand (i.e. aliases)
- Canned command procedures
- User-created command procedures
- Access to Menus a command

Relating commands to their functions is indirectly supported by a command language by allowing meaningful identifiers to be created. Command procedures, which allow a complex sequence of commands to be invoked with a single name, also provide brevity in communication with the system.

Object Oriented

Traditional command languages have presented the user with a monolithic structure of commands, normally organized only within a CL reference manual. The system has been viewed as a single object with a large set of diverse operations applied to it. This is the degenerative case of data abstraction. A system is actually a collection of objects. Operations within the system may be grouped by the objects they act upon. This grouping of operations and objects is equivalent to data abstraction, the object becoming abstracted and the operations becoming the only means to access the object. Modern language concepts can directly describe this view of data abstraction. The advantages of viewing a system as a collection of objects include:

- A more understandable system structure
- A clearer understanding of the system and its components
- Improved readability of the command language source that implements objects and operations

Command languages should similarly support creating the logic for controlling system functions as an object-oriented activity. Objects, such as files, directories and databases, are easily manipulated when viewed in this manner. A command language should reinforce a user's view of the system as a collection of objects, which have a set of operations that may be applied to them.

Problem Solving

The user of a system needs a command language that supports problem solving and expressing solutions to problems, using operating system services and application programs. The command language must support these qualities by allowing invocation of functions and analysis of the results of actions within the system. The language definition should include control constructs and variables that may be used in control constructs. The solution to a problem should be expressed in a concise manner within the language. Since a command language must support problem solving and expressing solutions, it should incorporate programming language features. These features include conditional branching, looping, argument handling, variables, string manipulation, expressions and exception handling, among others. The difference between a programming language and a command language should be minimized. When the same language is used for both programming activities and system interaction, the difference between programs and commands diminishes.

Organizational Structure

The command language definition should support the organizational structure of the users of a system. This implies support, within the language, of the ability to select a command in a user's environment that may be redundantly named with other commands. Normal programming language scoping is not sufficient to allow several versions of a command to be named the same and still select the version requested by the user. Supporting an organizational structure allows a single copy of a command to be placed in a library system, as opposed to each user or group of users replicating the command in their own libraries. Section 1.3 on the command language interpreter expands this concept.

Rehosting

Rehosting of a command language implies that it be independent from any specific operating system. Concepts unique to a specific command language or operating system present problems to a truly rehostable command language. Examples such a the UNIX piping or full-duplex communication refer to specific features that may be difficult to emulate or create on different computer systems.

Summary

Command languages must provide expressive power for an experienced user solving complex problems by using the system functions. At the same time a command language must be easy for occasional users to use and must provide many means to abbreviate the amount of information that a user must enter.

The command language should reinforce the concepts of object-oriented actions between the user and the system and should offer a consistent interface, preferably by using a single language to solve programming and system problems.

The organization of the users should be supported by the language and the language should be host-system independent, allowing a single user interface to exist regardless of the particular computer system.

1.1.2 Requirements for a Command Language Interpreter

This section states the requirements for a general user interface that are within the domain of the command language interpreter and are not part of the command language definition. A command language interpreter(CLI) is the program that accepts a statement in a command language, interprets the statement and effects a result in the operating system. As such, it offers many features to the user of a system that are not part of the language definition for a command language. A CLI is used to interpret both interactive and batch commands. The requirements for a CLI are listed in table 4.

Reduction of Keystrokes

Many of the features that a CLI implements are also partially offered by the command language definition. An example is the reduction of information that a user must enter. The CL definition may allow synonyms (as in Ada's renames clause), while the CLI may allow an unambiguous abbreviation to be expanded to the full name. The CLI may also reduce the number of keystrokes necessary by automatically completing some syntactic details, such as allowing spaces between parameters (by creating commas in place of the spaces). An important aspect of such actions by a CLI is that, to the user, the commands entered <u>appear to be in a different language</u>. This fosters confusion, since the language written in canned command procedures appears to be different from the abbreviated form allowed in interactive communication with the system.

Error Recovery

The CLI is responsible for error recovery. Error recovery should be designed to reduce the portion of the erroneous command that the user must reenter. Reentering a single parameter, name or value, or assumptions made by the CLI to correct the error (e.g. missing closing parenthesis) should be part of the error recovery. Error reporting is used by the error recovery and should be tailored to the experience level of the user and the mode of entry (i.e. line-by-line or full screen).

Prompting

Prompting is a function provided by the CLI. Prompting completes the command text by inserting information from the user. The CLI must be able to associate prompting with specific points in the command language definition, such as

TABLE 4

General Requirements for a Command Language Interpreter

- Reduce the keystrokes required for command entry
- Provide excellent error recovery
- Provide prompting for missing or erroneous information
- Provide helps
- Provide environments for retaining information relevant to commands
- Allow definition of user experience level
- Allow mode of entry to be determined
- Provide programmable function keys
- Provide an interrupt function
- Provide access to a menu system
- Allow both compilation and interpretation of command procedures
- Allow background and foreground execution of commands
- Provide searching for versions of commands not allowed within the command language definition
- Provide security and access control to commands and system objects
- Offer the features of the CLI to programs (e.g. prompting).

after the command name, after the opening parenthesis of the parameter list, after a parameter name, etc. The CLI must also be able to display standard and custom prompting information for each break point, allowing the possibility that a prompt for a specific parameter value could display a useful default.

Multiple Environments

The CLI should provide several environments for retaining information relevant to the command language. This information typically consists of default parameter values, definitions of the experience level of the user and preferred methods of system usage (prompting, etc.) as well as terminal characteristics. The environments should be defined as system-wide, user-specific and session-specific. The user-specific environment should retain information between sessions, while the session environment should be discarded upon logging off the system.

User Experience Level

The experience level of the user should be defined in the user and session environments. The experience level should qualify the interaction with the CLI, as in the areas of prompting and helps. Furthermore, the experience level should be modifiable on an individual command basis, allowing an experienced user to revert to an inexperience level for a new command.

Mode of Entry

The CLI should be aware of the mode of entry, whether the command is arriving from a full screen terminal or line-by-line terminal, e.g. hard copy. Mode of entry should also denote whether the command is interactive or batch and will qualify the actions of the CLI.

Programmable Function Keys

The CLI should provide for programmable function (PF) keys, which allow single key entry of commands or any other information that could be entered via the keyboard. The values represented by the PF keys should be stored in the system, permanent user or session environments. The session environment allows reassignment of PF keys that revert to their permanent assignments after end of session.

Helps

The CLI should provide helps on a command, parameter list, parameter name and parameter value basis. Helps should be tailored to the experience level of the user and to mode of entry (line-by-line or full screen).

Access to a Menu System

The CLI should provide for access to a full screen menu system. The menu system is a separate facility of a user interface and not part of the CLI. The CLI should allow invocation of the menu system and offer the facilities of the CLI to the menu system (e.g. prompting, helps, etc.).

Compilation and Interpretation of Command Procedures

Command procedures have traditionally been handled interpretively, usually because a command language was viewed as a high level means of referencing operating system functions and not as a programming language. When a common language is used as both the command language and the programming language, the difference between interpretation and compilation is minimized. The only difference between compilation and interpretation is in the execution time of the command procedure. Compilation and interpretation must be interchangable and guarantee the same results for the command.

Background and Foreground Execution

Operating systems have traditionally provided two modes of execution for commands: foreground and background. Background has, in many cases, involved using a completely different language from foreground communication with the system. When a common command language is used for both background and foreground jobs, the difference between the two modes is minimized. The differences between foreground and background execution of a command are:

- Background tasks have a lower priority than foreground, foreground implies that a user is waiting for the results of the command.

- Background and foreground tasks differ in the actions taken by the CLI when errors or prompting occur. Background tasks normally abort when user-supplied information is needed, however, they could be suspended if the user is logged on to the system while information is obtained from the user.

Organizational Structure

The organizational structure of the users must be supported by both the command language and the interpreter. Section 1.1.1 reviewed the requirement placed on the CL definition for this feature. The CLI must support searching for versions of commands that would not be possible within the CL definition. Multiple versions of commands typically are resolved by specifying a set of command libraries in a predefined order. The first command found is the one used by the CLI. This is different from the rules of scope that a command language may embody. Scoping does not allow for redundant names within the same scope, a feature that is necessary for multiple version of commands. The CLI must provide some mechanism for allowing an organization to specify the command libraries shared between users and projects. Providing a mechanism to search for commands does not, in itself, provide security or limit access to commands, as explained below.

Security and Access Control

Security to the system that interfaces with the CLI is provided by the CLI placing access controls on each command and system object. Users of the system have access rights defined that are used to verify the validity for each command and object referenced. The CLI should provide for this feature and also provide a reporting mechanism for violations to the system security. Access control should be hierarchical, which states that if a user is authorized to reference objects or operations at a given level, all operations and objects which require lower level access rights are automatically available to the user. Note that while a command may be visible to a user, through the search order provided by the CLI, it may still be unaccessable due to the security level of the user and the command.

CLI Features as Program Services

The standard features of a command language interpreter, which include error checking, error reporting, prompting and helps, should not be restricted to the CLI but should also be made available to programs and command procedures. This implies that the routines needed to perform error checking (e.g. semantic checking) and those needed to report and display the errors (e.g. error log) are also available to application programs that may choose to report errors in the same manner as the CLI.

Summary

The command language interpreter completes the user interface for a system by embellishing a command language with features outside of the CL definition. These include error recovery, prompting, helps and PF keys. The CLI allows user communication outside of the CL definition by providing an access to a menu system. The CLI provides for multiple permanent and temporary environments and for definition of user and terminal characteristics. Additionally, the CLI provides non-language features that include a choice of compiling or interpreting commands, a choice of command execution priority (background or foreground) and an organization to command versions, augmented by access controls to each command and system object. Finally, the CLI permits powerful application programs to be developed by offering its services as linkable, callable subroutines.

1.1.3 Analysis of Ada as a Command Language

This section evaluates Ada as a command language and identifies areas where Ada clearly supports the command language requirements discussed in section 1.1.1 and where Ada has deficiencies with respect to command languages. A brief overview of Ada's position in programming and command language is given, followed by two sections on Ada's ability and inability to satisfy the requirements for a command language.

A New Generation of Command Languages

Command languages have paralleled the development of programming languages. IBM's OS JCL parallels assembler language, requiring the user to be intimately familiar with system details. The IBM TSO interactive command language resembles Fortran, allowing primitive control constructs and global data. Burrough's work flow language allows block structure similar to Algol's. Current command languages have yet to incorporate many features of modern programming languages.

The newest programming languages are characterized by their ability to express data abstraction and allow object-oriented design. The emphasis is on defining the objects to be manipulated by the program, rather that emphasizing the statements that implement the manipulation of objects. Ada promotes this ideal, which is abscent in older programming languages, through the concepts of packages and private data. Operations on objects are conveniently collected within the same package as the object. Ada, as a language, allows the command languages to progress to the same level as modern programming languages. This, in turn, allows a user of a computer system to solve system problems by applying object-oriented design techniques.

1.1.3.1 Command Language Requirements Met by Ada

Ada's satisfaction of the requirements for a command language listed in section 1.1.1 are listed in table 5.

TABLE 5

Ada's Fulfillment of Command Language Requirements

General CL Requirement	Ada's Realization
Present Uniform Interface	Command Language is same as programming language
Be Easy to Use	Permits renames, default parameters and other ease-of-use constructs
Support Problem Solving	Supports complex problem solving including tasking, data abstraction and other difficult problems
Support Object-oriented Viewpoint	Directly, through the use of packages that contain private data types and associated operations
Support Multiple Versions of a Command	Indirectly, through normal rules of scope, implies that different versions are always in different scopes
Support Rehosting of CL	Directly, through the separation of independent specifications and dependent bodies for packages and tasks.

<u>Uniform Environment</u>

Using Ada as a command language satisfies the requirements for a uniform user interface with the system, since the language for program development and system communication are the same. Programming language features that are a requirement for a command language, such as argument handling, etc., are all satisfied by Ada, since Ada is first and foremost a programming language. Using Ada as the CL reduces the amount of learning that a user must undergo since the command language is the same as the programming language that is used.

Ease of Use

Ada supports an easy to use command language by allowing default parameter values, synonyms for command procedures, user-written command procedures and full expression of control constructs to direct the sequence of actions within a command procedure.

Problem Solving

Ada, as a programming language, naturally supports problem solving and expressing solutions to problems. Ada does more in this area than traditional programming languages, since it reinforces solving problems by analysis of the objects within the problem. Ada also directly supports data abstraction through packages and private data types. As a command language, Ada offers more solutions to solving problems than present command languages, which often are restrictive even in basic areas such as control constructs.

Compilation and Interpretation of Command Procedures

The issue of interpretation vs. compilation and the distinction between program and command source are minimized by using the same language for program and command development.

Rehosting

Ada directly supports independence from the host operating system for a command language by allowing the OS-specific portions of the command language to be localized in Ada packages. This allows recoding of the system-specific code to be easily identified, and supports a static interface to the rest of the command language.

1.1.3.2 Command Language Requirements Not Met by Ada

TABLE 6

Command Language Requirements not Fulfilled by Ada

General CL Requirement	Ada's Restriction
Ease of Use	Ada syntax and semantics must be followed, unambiguous abbreviations of Ada names not supported, extra keystrokes needed for parenthesis around parameter, semicolon for command termination, declaration of new object before use, other features assumed by some command languages.
Support Multiple Versions of a Command	Cannot support multiple versions (i.e. same name and parameters) in the same scope.

While Ada satisfies almost all requirements of a command language that are in the domain of the language itself, it provides only a static solution to the problem of command languages. The DCP command language must offer a dynamic solution to the problem of a user interface. Table 6 summarizes some of the features of a command language not support by Ada, and to which solutions may need to be offered by the command language interpreter, if these features are determined to be of more importance than using ANSI/MIL STD Ada as a command language.

The issues of dynamic typing versus static typing and dynamic elaboration versus Ada's imposed ordering impact the definition of Ada as used for an interactive command language. Ada, as a command language, is impacted in the areas of undefined objects, changing the environment of a user session and other nuances of an interactive command langauge. As an example, creation of new files (without prior declaration of the file) is a common feature of many command languages. Ada, strictly interpreted, requires the explicit declaration of the file prior to an operation. The DCP CL must support a program (i.e. the Ada procedure that is the user session) changing dynamically. This conflicts with some basic concepts in Ada, notably the placement of variable declarations, initiation of tasks, placement of exception handling and session termination. These issues must be addressed in using Ada as a command language.

1.1.3.3 Modifications to ANSI/MIL STD for the Ada Command Language

TABLE 7

Modifications to the MIL STD Ada for Use as a Cmd Lang

none as of this issue

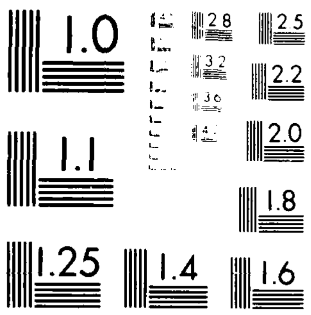
1.1.4 <u>Analysis of an Ada Command Language Interpreter</u>

The Ada Command Language Interpreter (ACLI) must support the requirements for a general command language interpreter outlined in section 1.1.2 while using a subset of standard Ada. Extensions to Ada The ACLI will fulfill the requirements stated in section 1.1.2, with the following clarifying points:

Note that this section describes only the ACLI, two related user-interface systems (the menu system and the Ada Preprocessor) are described in separate models and are distinct from the ACLI. The ACLI provides for interpretation of commands and procedures that conform to standard Ada, the Ada preprocessor provides an abbreviated, simplistic interactive interface to the user and generates correct Ada. The menu system, when interacting with the ACLI, also generates correct Ada. The ACLI is concerned with the ACLI system envi-

AD-A142 570 WIS IMPLEMENTATION STUDY REPORT VOLUME 3 BACKGROUND INFORMATION(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T H PROBERT 01 OCT 83 IDA-D-51-VOL-3 4/6

UNCLASSIFIED IDA/HQ-84-28344 MDA903-79-C-0018 F/G 17/2 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

ronment, the interpretation and execution of commands and the synchronization of system responses to the user. Menu systems are dealt with in section 1.4. The Ada preprocessor model is in section &secpre..

Error Recovery

The ACLI will support error recovery by applying an interpretive technique that is sufficient to make both assumptions about as many missing syntactic details as possible and to attempt a recovery by inserting new information from the user. An example of insertion of syntactic elements is adding missing commas between parameters, terminating parenthesis for a parameter list, etc. Error recovery must be able to introduce new information, supplied by the user when prompted, into the original command and attempt a successful parse of the command. Error recovery must therefore function in an interactive mode and be sufficient to operate in batch mode without additional input.

Prompting

Prompting requires that the ACLI be capable of associating points within the command language definition where prompting is supported. These points should include, at a minimum, prompting for parameter lists, individual parameters, parameter values and confirmation of certain critical commands prior to invocation (e.g. delete file). Prompting must interface with a database that supplies the command-unique information to be displayed to the user, e.g. the name of an individual parameter. Prompting must also allow the user to get help on the specific problem discovered in or associated with the input. This implies that prompting for a single parameter name would reduce to help for only that parameter, if the user requests help instead of supplying the parameter.

Helps

The ACLI must also provide access to helps for commands. Helps should be offered either as line-by-line descriptions or as a full screen display of the same listing, possibly reformatted to take advantage of the full screen capabilities. Helps have several forms:

- A help command that displays all, or part of, a description for a single command

- A help that briefly presents specific information for a component of a command (e.g. parameter)

- A tutorial describing a system accessed by one or more commands.

Part of the function of the ACLI is to provide access to helps and tutorials in a predetermined fashion. Helps should be able to be traversed, with each lower level of help providing more detailed information. A table of contents and index for a set of helps relating to a command should be available, as an alternative method to traversal for recalling helps. The help facilities should be invocable through either commands or a special programmable function key.

Programmable Function Keys

Programmable function (PF) keys allow a user to invoke a a system function by pressing a single key. Programmable function keys should allow any key on the terminal keyboard to be defined as a function.

A default set of functions should be assigned to the terminal keys by the ACLI, yet be modifiable by the user. It may be advantageous for the ACLI to require that a set of required functions always be present (i.e. assigned to some key), such as the help PF key, yet allow redefinition of these functions to different keys.

Programmable function keys should be capable of representing any string of characters that may be entered from the keyboard, without regard to whether they are correct Ada commands. PF keys may be used for program data, concatenated Ada commands, parameter names and/or values, etc. Non-printable characters, such as line feed and carriage return should also be representable within the character string for a PF key.

A mechanism must be present to allow interpretation of normal keys as program function keys. The translation of the physical terminal input to logical PF key should be buffered from the ACLI through the use of a rehostable I/O package that defines all program function keys in high level language terms.

Programmable function key definitions, like most other environmental data, should be kept on a session, user and system basis. It should be left to the user to save the redefinitions of PF keys in either the user's permanent profile or leave the definitions to be discarded at the end of the session.

Compilation vs Interpretation

The ACLI is principally responsible for minimizing the difference between compilation and interpretation of command procedures. By supporting more than only an interpretive form of a command procedure, the ACLI permits executable load modules to be referenced in the same manner that interpretive modules are referenced. Thus, to the user, the difference between executing a compiled command procedure and interpreting a command procedure are transparent. The user will only notice the increase in response time for executing a compiled command procedure. The ACLI's support of compiled and interpreted command procedures results in the requirement that it handle multiple forms of executable modules.

Organizational Structure

The organizational structure of users within the system is supported by the ACLI through a mechanism for selecting versions of commands. Versioning of commands is outside the scope of the command language definition. The ACLI supports versioning by interfacing with a configuration management tool that allows the proper version of a command to be used. The definition of "proper version" is a function of the project management that defines the groups of users of the DCP system. The ACLI provides the possibility of allowing different versions to be referenced by different users.

Security and Access Control

Security and access control are managed by the ACLI. Security should relate a user's capabilities granted within the system to commands and system objects. This implies that each command and object in the system will have a security level attached to it, while every user has a level of access associated with himself.

1.2 ADA COMMAND LANGUAGE MODEL

The DCP will use Ada as a command language to provide a uniform portable user environment across the various DCP hosts. The use of Ada both as a command and implementation language reduces the number of languages that must be understood by the DCP user. Ada is particularly well suited as a command language because of its packaging and abstraction capabilities.

A model is used to explain the use of Ada as a command language. A model is a representation of a system, using analogies to help visualize the system. A model is useful for understanding a system at a conceptual level and for verifying the postulates about the system prior to a functional description of the system.

This section describes what is means to use Ada as a command language without regard to how the command language interpreter is implemented. This command language model is important in that it defines the user view of the DCP command language interface. The model will drive the functional description of the Ada Command Language Interpreter (ACLI).

This section presents only the model of the Ada command language. Models for the Ada command language interpreter and for a user session are presented in sections 1.3 and 1.5 respectively. The collection of the three models fully describes the use of Ada as a command language within the DCP system.

1.2.1 Simple Commands

Ada, as a command language, is the primary user interface to the DCP. As such, it must allow simple operations to be performed easily and efficiently without the overhead and complexity needed to support sophisticated operations. In the simplest case, a command is a single line invocation of a system function or user application.

Figure 1 gives an example of simple user commands to invoke standard system utilities. The Ada commands are simply procedure calls to system utility functions or to user defined programs. Arguments used by the utilities are passed as normal Ada parameters. The use of Ada as a total programming environment is reinforced through the idea that the user is an Ada procedure, or task, active in the system. The user's procedure defines a session with the system and may invoke other procedures. The invocable procedures are all viewed (as an interface) as Ada command procedures, although the actual executable object may be a program, command procedure or system function.

Ada allows invocation of system functions by entering a simple Ada statement. Arguments to system functions are handled in a straight forward manner as parameters, implying the ability to define arguments by name and type.

In the Ada command language, there are no assumed defaults for any parameters, unlike some command languages such as UNIX which provide implicit defaults for pipelining. Ada requires that all default values be stated in the Ada source declaring the parameters. Coding standards will be necessary if command procedures need to share the same default parameters. By convention, a default parameter value may be assigned to parameters within Ada command procedures, resulting in a uniform default environment to the user.

An important area of departure from strict Ada semantics occurs for parameter values that are not defined at time of invocation of a function, but rather are returned values from the function itself. This is illustrated in figure 1, where the copy function could be defined as creating the "TO_FILE" if it does not already exist. This implies that the value for the parameter would not exist as an object when the copy function is invoked. The significant difference between instantiating an object (through an Ada declaration) and typing the name of the object should be noted. In figure 1, the name is simply typed and no instantiation has been done using an Ada statement. A strict interpretation of Ada could not support the semantics of such a function. If undefined objects (such as the file in figure 1) are allowed, the command language interpreter will need to support the semantics, since the Ada language cannot.

```
-- Following are standard file utilities

COPY(FROM_FILE, TO_FILE);
DELETE(FILE);
RENAME(OLDFILE,NEWFILE);
   {etc.}

-- The following command runs an
-- application program which reads data from the
-- 'DATAFILE' and produce results in the 'RESULTFILE'.

ANALYZE(DATAFILE, RESULTFILE);


-- The following invocation of ANALYZE would write the
-- results to the terminal.

ANALYZE(DATAFILE);
```

Figure 1: Examples of simple operations - Ada procedures

1.2.2 Sequencing Commands

More complex examples can be created by sequencing calls to commands. Control of the sequence may be done by the user through the use of Ada control structures or the creation of data variables. In figure 2, a file function "EMPTY_FILE" is used to query the contents of a file created by a previous step. Commands may be entered individually or sequenced through the use of normal programming constructs. Ada allows a consistent, simple interface between the user and the system and requires that the user be knowledgable in only one language.

```
ANALYZE(DATAFILE,RESULTFILE);
if not EMPTY_FILE(RESULTFILE) then
   PRINTOFF(RESULTFILE);
end if;

-- The 'RESULTFILE' is printed only if
-- it is not empty.
```

Figure 2: Example of an Application Function

1.2.3 Command Procedures

It is often desirable to perform a sequence of commands in an abbreviated manner. Sequences of commands may be created and stored for future invocation, as a group. In the ACL, these groups of commands are simply Ada procedures. Ada procedures and command procedures are equivalent. In the ACL, invoking a command (or command procedure) is equivalent to calling an Ada program. Command procedures can be defined to allow a user to reuse a sequence of commands. The command procedure allows the programmer to present the application user with a much simpler interface. Ada procedures allow several capabilities desirable when creating a sequence of commands:

- Commands may be grouped and invoked with a single name (the command procedure name).
- Arguments can be passed to the command procedure (in the form of parameters).
- Information may be retained during the execution of the command procedure, normally to control the sequencing of commands (in the form of local variables). Variables, in the ACL, have properties of scope that permit great flexibility in a command language. The outer most scope is that of the ACLI, which contains predefined variables shared by all users of the ACL. An inner scope is defined by the Ada procedure that is the user's session. Variables within this scope have a lifetime of the session. Command procedures created within the session may have their own variables, providing scope that has a lifetime only during execution of the command.

- Other command procedures, commands or programs may be used from sets of operations granted to the user, e.g. file operations, (through the WITH and USE statements).

- Interrupts during the execution of a sequence of commands may be easily handled (using the Exception statement). Interrupts follow the Ada rules of propogation. Command procedures are nested, with the outermost procedure being the ACLI itself. Within the user's session procedure, many command procedures may be activated. Interrupts are handled by aborting the active command procedure, upward, until one is found that specifically handles the interrupt. The ACLI handles all interrupts, allowing an abort message to be displayed if the interrupt has not already been handled. As defined in the Ada language reference manual, handling of an interrupt disables the interrupt handling mechanism.

- Command procedures maybe used to launch concurrent tasks, such as a background compilation step, (through the use of Ada tasking). All tasks are launched under the user's session task, resulting in the requirement that the ACLI handle background tasks still executing when the user's session task is ended in an orderly fashion. Tasks may rendezvous, allowing the possibility of background tasks requesting input from the user's session task through a rendezvous point. Complications, such as the user not being logged on and therefore unable to rendezvous, must be handled by the command language interpreter.

The use of command procedures is illustrated in figure 3. The command procedure necessary for an operation is illustrated.

Command procedures permit complex interactions to the system to be created and referenced in an abbreviated manner. Control is granted to the user through the use of several Ada constructs, including variables and types local to a command procedure.

```
with TERMINAL_IO; use TERMINAL_IO;
procedure LIST_DIRECTORY (ROOT : in DIRECTORY) IS

  -- This command procedure will list a specified directory. The
  -- root determines which directory is listed.

  I, N :  INTEGER;

begin

   N := NUMBER_ENTRIES(DIRECTORY);

   if N <> 0 then
      DISPLAY('DIRECTORY LISTING');
      for I := 1 to N
        loop
          DISPLAY('> ',GET_ENTRY(DIRECTORY,I));
        end loop;
   else
      DISPLAY('DIRECTORY EMPTY');
   end if;
end LIST_DIRECTORY;
```

Figure 3: Example of a Command Procedure

1.2.4 Command Packages

The packaging of command procedures allows logical grouping of functions to be expressed using the Ada package construct.

Command procedures are normally grouped by functional area, as frequently seen in command language reference manuals. This is a natural occurence of the fact that commands act upon objects in the system. For example, commands that copy, delete and create files all deal with a file object. To the user, this provides a simple and convenient means of understanding the interaction with the system.

Ada supports this natural grouping of operations through the package concept. In Ada, a package provides a means of grouping several operations together and presenting them to the user. A "file package" may fully define all of the operations that a user may perform on files in the system. Ada packages do more than simply group operations, they also allow two important concepts:

- Ada packages define, in a concise manner, the object that the user is manipulating. For example, in a file package, not only are a list of file operations given to the user, but the fact that all of these operations act upon a file object is clearly stated in the Ada Package construct.

- Ada packages allow use of common (global to the package) information. This includes variables, types and constants. Thus, a file package could easily define a "file name" type that all operations within the package could reference. The information in a package also has the capability of being available to the user of the package or hidden within the body of the package. This allows packages to describe more than operations: objects and descriptions relevant to objects may also be described.

In the DCP system, packages may be viewed as belonging to one of three classes:

- System Packages

  System packages provide standard operations on standard objects within the system. These are typically files, directories, time and date, etc. It should be noted that in the ACL, <u>no DCP system functions are predefined</u>. The ACL provides a framework for defining packages. This framework allows independence from the host system and defines a high level description of the system interfaces.

- User Packages

  User packages describe a user of the ACLI system. This description includes security information, allowable functions and other customized information. The user himself may modify a portion of this package to create abbreviations for commands, new command procedures and any other capabilities made possible through the use of an Ada package. Ada packages provide a convenient method for describing a user to the system.

- Other User Packages.

  The ACL supports the organizational structure of the users of a system through the USE and WITH statements. These statements allow selecting specific packages of commands, either explicitly (e.g. package.command) or implicitly (with the USE statement).

1.2.5 Command Tasks

Definition

A task is one of Ada's three primary program units, the other two being subprograms and packages. Packages have been previously discussed and are used primarily for organizing source and concisely expressing data abstraction. Subprograms are equivalent to procedures and have also been reviewed.

Concurrency

Tasks are used to express concurrency among a number of activities. In an operating system, concurrent activities consist of the users logged on to the system and certain system-resident tasks, shared by all users (e.g. time of day). A task is simply an entity that operates in parallel with other program units. Within this context, a user session is viewed as an activity and therefore a task. Thus, user tasks operate in parallel with other user tasks and system-resident tasks.

Review of Task Features

Ada's definition of tasks provides for several features relevant to a user's session with a system:

- Tasks may communicate with each other, via messages.
- Tasks may define multiple entry points for other tasks to invoke.
- Each entry point has an implicit queue associated with it, providing an automatic mechanism for output spooling, even multi-level output spooling.
- Tasks, not being separately compilable, must be enclosed in packages or subprograms. This is ideal for packaging all users of a system under a single "system_users" package. Packaging provides a convenient means for documenting the users of a system and allows shared data to be concisely declared. Other implications of this type of packaging are recompiling the ACLI to add new users to the system and viewing each user as a dormant task in the ACLI, ready to rendezvous with any task operating a terminal (permitting multiple logons by a single user).

- Tasks may rendezvous, providing an automatic means of synchronization between concurrent tasks.

- Ada tasks provide for "families of entities", tasks which are indexed by some value and are, in effect, an array of similar tasks.

- Tasks may be instantiated dynamically, allowing one task to spawn several other tasks.

- Tasks may choose to be suspended by either a busy wait, which uses processor time and resources, or by a sleeping wait, which suspends a processor without using resources.

- Tasks may instigate a delay, and choose to abort an attempted rendezvous after the delay has been met.

Having briefly summarized some of the features of Ada tasking, an examination of how tasks fit within the Ada command language model follows below.

Within the command language, users will be considered to be tasks. This provides for viewing the user as a concurrent activity within the system. Entry points within the user task will be defined to accept output from system services and applications that the user invokes. Multiple entry points will probably be found to be useful by allowing the user to have more immediate access to system responses than to output generated by application programs. Tasking allows output directed to the user to be queued until the user task is active and rendezvouses with the entry point queuing the output.

The user task will be a task type, defined by the ACL and instantiated once per user. A single instantiation still permits multiple logons, since a logon is considered to be a terminal task rendezvousing with the user task. The instantiation will be within a single package declaring all of the user tasks possible for a system. The ACLI use of the user task is explained in section 1.3 and is not a concern for the command language model.

Figure 4 gives a sample of what the user task type might look like. Figure 5 shows a hypothetical system user package that contains a set of valid users for the system.

The user tasks may enqueue output from application programs while the user is either entering new commands or accepting output from another application.

```
TASK TYPE   user_task    IS
  ENTRY response(msg : IN sys_response)     ;
  ENTRY output  (msg : IN output_type)      ;
  ENTRY communication (msg : IN input_type) ;
END user_task;
```

Figure 4: User Task Type for the ACL

```
PACKAGE system_users IS
   first_user  : user_task;
   second_user : user_task;
        .
        .
        .
   last_user   : user_task;
END system_users;
PACKAGE BODY system_users IS
        .
        .
        .
  TASK BODY first_user IS
    ACCEPT output(msg : IN output_type) DO
      END output;
    ACCEPT response(msg : IN sys_response) DO
      END response;
    ACCEPT communication(msg : IN input_type) DO
      END communication;
  END first_user;

END system_users;
```

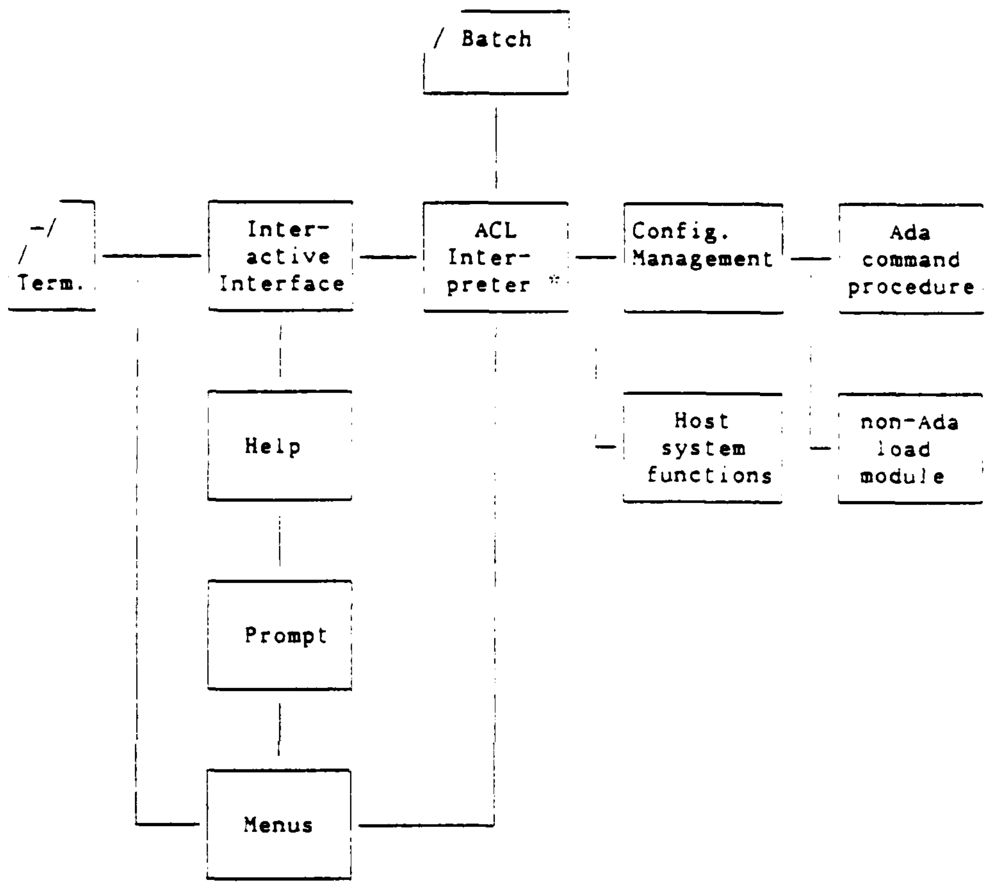
Figure 5: Sample System Users Package

1.3 COMMAND LANGUAGE INTERPRETER MODEL

The ACLI model consists of three main parts:

- An interpreter that interprets complete, correct Ada source and executes (directly or indirectly) the system functions referenced in commands.
- A configuration management interface that allows selection of versions of commands, permitting comr ıds to be tailored to a group of users.
- An interactive part that completes Ada text from incompiete user input. The interactive part provides for prompting, abbreviations to components of Ada statements (e.g. abbreviated names) and all other modifications to the Ada language definition tha enhance an interactive user interface.

The relationship between the interpreter, interactive interface, configuration management and supporting tools (helps, prompting, menus) is illustrated in figure 6.



* Note:
contains builtin DCP functions

Figure 6: ACLI System Model

1.3.1 Interpreter Model

The interpreter portion of the ACLI interprets Ada source code, invokes DCP system functions and provides the services of the ACLI for supporting an Ada command environment. The interpreter will accept only syntactically and semantically correct Ada code. The code may arrive from either an interactive user or in batch mode. The ACLI provides for comprehensive error detection and reporting, but not recovery. Error recovery is provided in the interactive mode by the interactive interface component of the ACLI.

The functions of the interpreter portion of the ACLI include:

- Translation of Ada statements to invoke commands and command procedures, to define objects and perform other actions necessary for problem solving.
- Provide for communication between tasks within the DCP system.
- Provide an interface between the user and the DCP system, via the user's input device (terminal or batch job output).
- Provide an interface to a configuration management system that will allow selection of versions of commands.
- Provide an interface to a menu system.

1.3.1.1 Commands and Command Procedures

The definition and executable form of operations referenced by a user is accessible by the ACLI. Referencing an operation results in two basic types of information being made available to the ACLI: interface information and executable form of the function.

The ACLI can be used to invoke functions written in Ada as well as other languages. This is made possible by describing the function in Ada, as an interface specification. The executable form of the function can be created by non-Ada compilers. Attaching an Ada interface to such software provides a consistent interface for all functions to the the ACLI. The executable form may be a load module or a command procedure in some intermediate form.

When the ACLI is invoked with the name of a function, it can verify the function's existence and semantic correctness of the parameters. Parameters are verified by name and by the type of the parameter value. Semantic checking results in the function interface conforming, by function name, parameter name and type of parameter value, to the Ada specification for the function. The ACLI then executes the function in one of a number of ways, as discussed below.

Figure 7 illustrates the relationship between commands and programs in the DCP. Executable load modules for user-written programs are produced by compiling Ada source through an Ada compiler. The load module is stored in the library. The program is not invocable using the command language until an ACL interface has been written. The interface is the Ada specification for the procedure. The ACLI frontend processes the interface specification and produces a form that is understandable by the ACLI backend. This form is stored in the library. Upon invocation of the program, the ACLI backend references the interface information and passes control to a loader for loading and executing the load module.

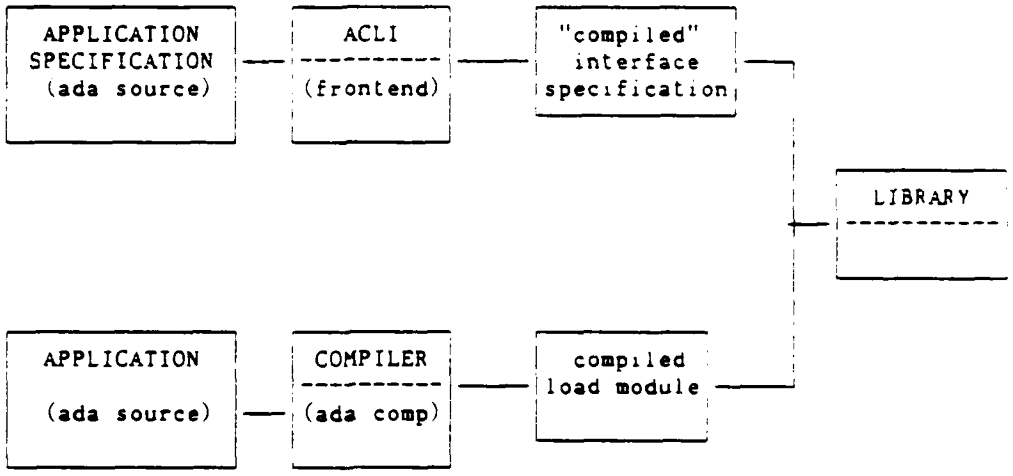


Figure 7: Specification of a User Function

1.3.1.2 Interface Information

Every system function supported by the ACLI has an Ada interface specification to which the command invocations must conform. This specification states useful information for the ACLI giving the function name, the parameter names and types and the type of the returned values (for the function). These interface specifications are defined in Ada packages that group operations with similar functional properties.

1.3.1.3 Executable Forms

Once the ACLI verifies the interface information, it executes the function. Executing the function may involve retrieving an intermediate representation of the command procedure (precompiled Ada command language source) and interpreting the statements within the procedure. If the function is a compiled program, a load module will need to be retrieved and executed. A third method of execution is for the ACLI t› call, directly, a resident host system function. These three forms of executable objects are described in the following points.

- Command Procedures

  Command procedures that have not been compiled will exist in interpretable form and be executed by the ACLI backend directly interpreting their intermediate form. This implies that the command procedure was written in Ada, pre-compiled by the ACLI frontend to create an intermediate form and stored for reference by the ACLI backend.

  In figure 8, the steps performed by the ACLI when executing an interpretable command are illustrated. The user enters a command, which is verified by the ACLI backend by referencing the command interface specification. The backend detects that this command is in an interpretable form (part of the interface specification) and retrieves the form from the library. It interprets the command and diplays the results with a system response to the user.

- Load Modules

  If the command references an executable load module, the ACLI will pass control to a loader, after verifying the existence of the function and checking the parameters, using the interface information. The ACLI can verify the semantics of the interface, passing the parameter values and name of the load module to the loader. Further checking on the part of the loader is host system dependent. Actual execution of the function is handled by the loader, which returns control to the ACLI at the end of execution.

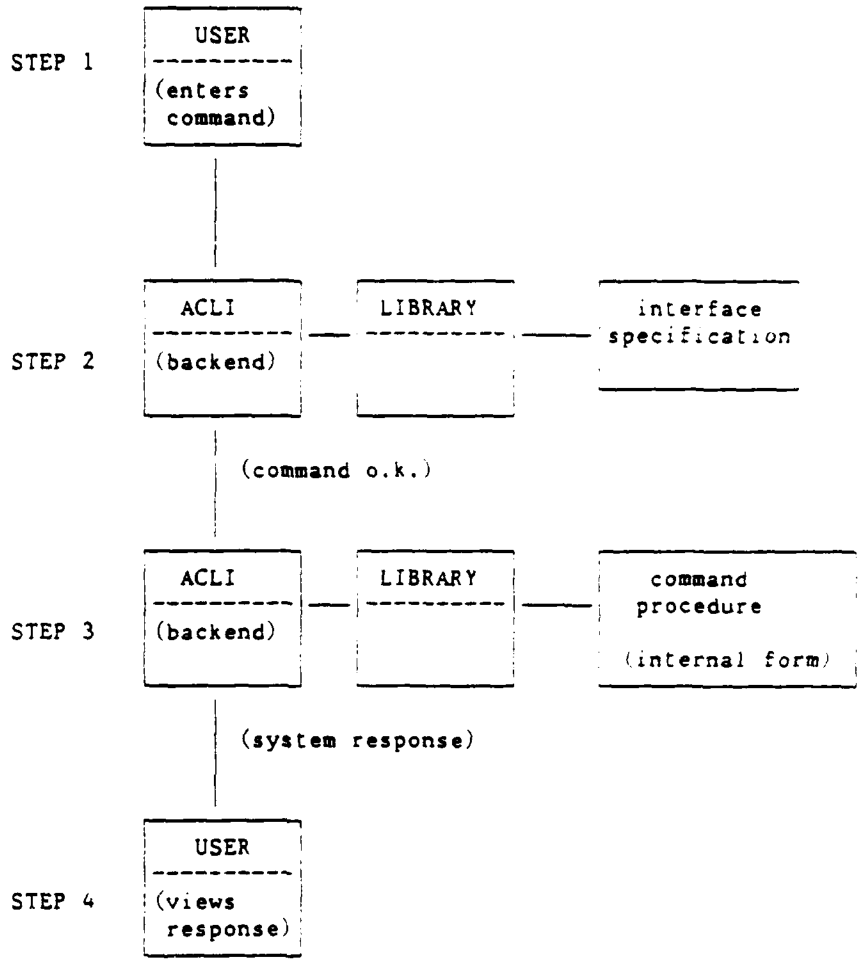


Figure 8: Execution of a Command Procedure

In figure 9, the ACLI interprets a command and detects that the command is in the executable format of a load module. The creation of the load module and the source language is not shown in the figure. After verification that the command is correct, the ACLI passes control to the system loader, with the name of the load module. The loader loads the load module executes the module and returns control to the ACLI. The ACLI finishes by displaying a system response message to user.

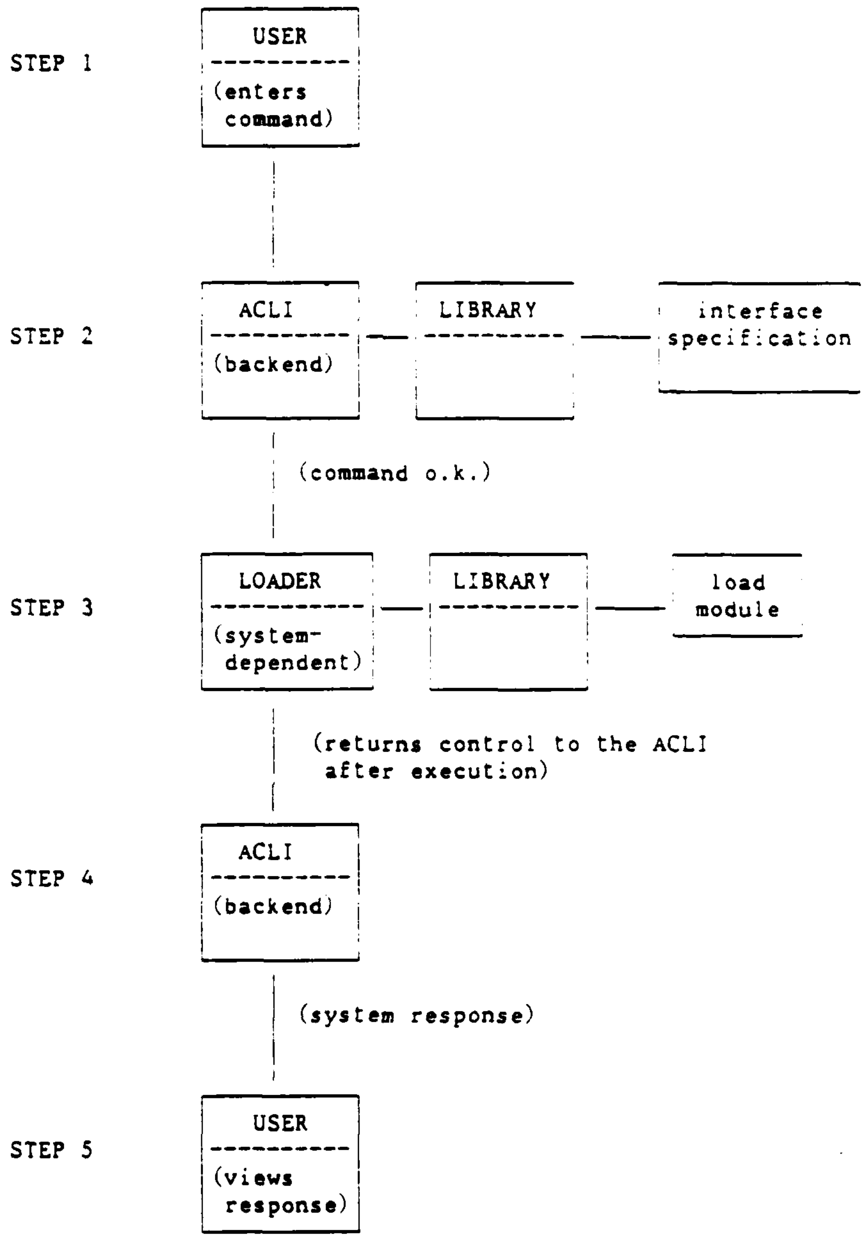


Figure 9: Execution of a Compiled Program

- System Functions

  As an optimization, the ACLI backend will contain some DCP system functions and a means of directly calling other DCP and host system functions without retrieving either a load module or a an interpretable form. This allows immediate execution of DCP and host system functions that are resident during interpretation of a command. The DCP system functions that are part of the ACLI will be subroutines, compiled with the ACLI and therefore part of the ACLI task. The ACLI will still need to verify the function interface using the library system.

  As illustrated in figure 10, the ACLI both verifies the existence of the command and executes the command by referencing code within its own task. The system function exists either as a subroutine or may be called directly as a system-resident utility.

1.3.1.4 Input/Output

Input and output between the user and the system is controlled either directly by the ACLI or by the screen manager. In cases where the terminal does not support a full screen interface, the screen manager simply passes I/O to the system-standard I/O package.

Use of a screen manager provides for a virtual interface to the ACLI and allows the screen functions to be separated from the ACLI itself.

I/O always reduces to communication with a standard I/O package that provides for direct and sequential I/O to system ports (e.g. terminals).

<u>Screen Manager</u>

The screen manager buffers I/O between the standard I/O package and the ACLI or application program. Buffering allows for the possibility of an individual screen per activity (or task) in the system. Thus, if the user has initiated several concurrent foreground tasks, each would communicate with a different logical screen. The screen manager would allow the user to select and dimension logical screens unto the single physical screen that is present in the terminal.

<u>Output Entry Point</u>

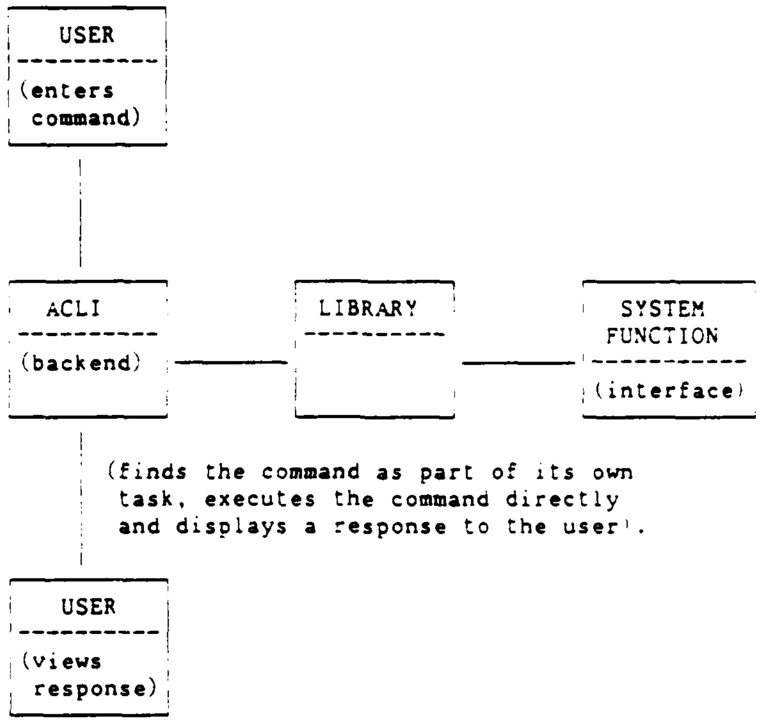

USER
(enters command)
ACLI
(backend)
LIBRARY
SYSTEM FUNCTION
(interface)
(finds the command as part of its own task, executes the command directly and displays a response to the user).
USER
(views response)

Figure 10: Execution of a System Function

Within the user task, an standard entry point termed "output" will be provided as a repository for all output generated by application programs. This is differentiated from the entry point termed "response", which allows system responses to arrive at a different queue from application output. The use of Ada entry points within tasks will be utilized to provide buffering and separation of types of input and output.

Response Entry Point

The response entry point provides for acceptance of DCP system response messages only. Response messages are queued at this entry point.

1.3.2 Configuration Management Model

1.3.3 Interactive Interface Model

The interactive interface portion of the ACLI provides an interface between the interpreter (which accepts only correct Ada) and the interactive user. The interactive interface is not used in batch mode and provides:

- Modifying the user's input text to create correct Ada.
- Access to helps, prompts and menus; all of which are unavailable in batch mode.
- Syntax checking, since the interactive interface creates syntactically correct Ada.
- Entry points for communication with the interpreter to intercept semantic errors and attempt excellent error recovery.

The Ada source that the interactive interface creates is accessible by the user, allowing further editing, copying and printing of the source. The user may choose to use the interactive interface as an abbreviated way of cerating Ada text. The interactive interface is conceptually a tool to ease creation of Ada text. It could be implemented as a syntax-directed editor, a collection of smaller user interface tools or in any manner that satisfies the concept of creating complete Ada text from a user-friendly interface.

1.3.3.1 Modifications to Ada

Handling of Ada Statements

The ACLI provides for the dynamic model of the Ada command language. A major part of this dynamic model allows the ACLI to insert Ada statements in the proper order within the user's task. Statements arrive chronologically from the user but may need to be placed either at the beginning of the user task, before the body of the task (begin block) or at the end of the task (e.g. exceptions). The ACLI provides for this by conceptually placing the statements in the proper order, as defined by Ada. Deletion of statements ("UNUSE", exceptions, declarations, etc.) is also permitted by the ACLI. This is permitted for the interactive mode of command entry only.

Syntax Enhancements

Other modifications to Ada include allowing a truncated, unambiguous name (for a function or parameter name), to be entered by the user and completed by the interactive interface. The interface will also supply innocous syntactic details such as enclosing parenthesis around parameter lists, commas between parameters and other syntax that may be safely assumed.

1.3.3.2 Helps

The interactive interface provides access to helps, either as a help command or through a help function key, which may be pressed at any point during command entry. The help that is selected will be determined by the context of the command when the key is pressed. The help command presents the user with complete or condensed information about a command.

The help key function is an integral part of the ACLI and does not have text associated with it, as do the other programmable function keys. Both help and interrupt keys request their functions without the necessity of interpreting text as an ACL command.

1.3.3.3 Prompting

Prompting will be provided for whenever text is entered that may be completed by prompting the user for additional information. Prompting may request entry of a list of parameters, a particular parameter or a particular parameter value. Prompting will also allow successive prompting of parameters, until the parameter list is complete. This will allow the user to choose prompting, in order to get a list of parameters and their default values, which may be entered or changed by the user.

1.3.3.4 Interrupt Handling

Interrupt handling must be provided for:

- Interrupting entry of the current command.
- Interrupting execution of the current command.
- Successive interrupting of queued commands.
- Fail-safe interrupt, which clears all user tasks and resets the user to an initial point of interaction with the ACLI. point

Interrupt handling will be provided, as is help, with an interrupt command(s) and keys. The interrupt key will be an implicit function, with no text associated with it.

1.3.3.5 Programmable Function Keys

The interactive interface will interpret programmable function keys to text. The text may represent Ada commands or any valid character string.

Some programmable function keys will result in direct action by the interactive interface, while others will be passed along to either programs, the ACLI interpreter part or other processes.

1.3.3.6 Menu Interface

The interactive interface will provide access to a menu system. Access will be via a command to the interface and will result in the interface invoking the menu system as a normal DCP system function (via the interpreter).

1.4 DCP MENU MODEL

The DCP system provides a menu interface to the ACLI. The menu interface allows Ada commands and program data to be entered via menus and displays menus containing system responses, helps, prompts and program data. The functions of the menu system allow command and data entry via a full screen by:

- Allowing system responses and other output to be displayed in a full screen.
- Permiting definition of menu frames for designers of helps, prompts, commands and programs.
- Providing the services of the menu system to Ada programs, as compilable entities.

1.4.1 Helps

The menu system provides full screen helps, using either the same helps that are used in line by line mode or customized helps for the full screen.

1.4.2 Tutorials

The menu system will provide access to a tutorial system. The tutorials will provide default services of:

- Displaying a table of contents for the tutorial
- Displaying an index for the tutorial
- Traversing the tutorial, to display successively more detailed information for the user

Tutorials differ from help by providing an on-line users guide that is browsable. Helps, on the other hand, are designed to provide concise, easily understood information for completing the command.

1.4.3 Command Menus

Command menus are created by the designers of Ada commands and command procedures. Command menus display a command name and list of parameter names and allow entry of the parameter values.

1.4.4 Data Menus

Data menus allow entry/display of an arbitrary screen full of data. Data menus are useful for communication between the user and programs.

1.4.5 Table Menus

Table menus allow entry/display of data between the user and a program, with builtin functions provided by the menu system. These functions include scrolling, selection of individual elements of the tables and other services.

1.5 ADA COMMAND LANGUAGE SESSION MODEL

1

1.5.1 Session Initiation

When Ada is used as a command language the user operates within the system as an Ada task from which system functions can be invoked. Initiating a session results in the ACLI rendezvousing with the a user task. The Ada source for the task is prefaced by appropiate WITH and USE statements to grant (to the user) functions, objects and abstract data types as defined by the system.

The use of Ada as both a command language and a programming language presents a uniform syntax to the user of the system. While this uniformity reduces the number of languages that a user must know, it increases the number of keystrokes needed to enter a command, since the command must conform to Ada syntax. Enclosing parameters in parenthesis, terminating a command with a semicolon and other details of the command language syntax are necessary to conform to Ada. The Ada preprocessor may be used in interactive mode to complete many of these syntactic details.

Figure 11 illustrates the initial sequence of a user session. Lines beginning with ">" indicate a user-supplied input to the system, lines preceded by ":" indicate a system response and lines preceded by "@" indicate commands issued by the system to the ACLI and not seen by the user.

Several items should be noted from the figure:

- The system has a single package that contains a task for each valid user of the system. The package, in figure 11, is named "logon".

1

DISCLAIMER

The session module contains some specific examples that may change after implementation. These examples are included solely to clarify the model and are subject to change in the final product.

```
> logon.userid;

  {userid is a task in the "logon"
   package, referencing the task
   activates it}

  {the userid task is an instantiating
   of a user_task type package.  This
   package includes the following
   statements, among others ... }

@ WITH system_env; USE system_env;
@ WITH userid_env;   USE userid_env;

  {the system responds by activating
   the task and displaying the message: }

: logon proceeding .....

>
```

Figure 11: Sample Session Initialization

- Logging a user on consists of the ACLI activating the task. The task itself is an instantiation of a generic task type. This task type contains all default WITH and USE statements, as well as an invocation of a procedure named after the user identification. The user procedure is invoked, which performs further customized commands prior to returning control to the ACLI.

- The first command that the user enters is actually the first statement of the user's Ada procedure. The user, executing under his Ada procedure, may alter statements that appear anywhere in the Ada procedure. WITH and USE statements may be changed to alter the user's environment. This will allow the user to select functions from different packages.

1.5.2 Command Entry

After initiating a session, the user is free to enter any command granted by the list of packages (WITH/USE clauses) preceding the user's (session) task. In figure 11, the system_env and user_env packages were accessible to the user, along with all of the operations contained in those packages.

Figure 12 shows some sample commands that the user might enter. The conventions for ">", ":" and "@" follow those in figure 11. The symbol "=" indicates a prompt for data input directly to a program and "#" indicates a display directly from a program.

```
> display(date,time);
: January 11, 1983 10:20.33
> line_editor.edit(temp.data);
# top of data
= change 10 'a' 'b'
# 10 A line with b in it
= end save
>
```

Figure 12: Sample System Interaction

In figure 12, the user's first command is a request to display the date and time. The ACLI invokes the "display" operation in the "date_and_time" package. Note that this assumes uniqueness of the name "display" and also that the "date_and_time" package is assumed to be contained in the "system_env" package referenced in figure 11 as a header to the user's session procedure (i.e. the "WITH system_env" clause). The date_and_time package responds with a line showing the date and time.

The user then invokes edit, in the example using an explicit package qualification, illustrating the possibility of switching between packages in the system. The "line_editor" package is assumed to be included in the "system_env" package but without a "USE" statement. This allows a default editor (e.g. screen editor) to be granted to the user while still having other editors available. The flexibility afforded in Ada with packages and access to packages (via the

WITH and USE statements) may be exploited to create a system or user package that contains more than the directly accessible operations.

The line editor takes control of the screen after invocation and issues the message "top of data". The user inputs a line editor command "change 10 ... ". Note that this is actually data to the line editor and need not conform to Ada syntax.

Finally, a return to the Ada environment is made with the data "end save", terminating the line editor operation and resulting in a prompt by the ACLI.

It should be noted that figure 12 is an example illustrating full qualification of a command and making assumptions about the form of commands to the editor. Figure 13 shows the same transaction in interactive mode, which allows abbreviated names and other techniques to save keystrokes. The correct Ada that the interactive interface generates is shown along side of the user input. The number of keystrokes saved is apparent.

```
USER INPUT                    GENERATED ADA
> ti                    time; {name abbreviated}
: 1/11/83 10:20.33
> ed temp               edit(temp.data);
                              {name abbreviated,
                               data set qualifier
                               assumed}
# 1 10
= c /a/b/
# 10 A line with b in it
= end
>
```

Figure 13: Sample Abbreviated System Interaction

1.5.3 Command Procedure Entry

The user may create command procedures. Command procedures are created using an editor, precompiled using the ACLI frontend and interpreted using the ACLI backend. Command procedures are invoked (after precompilation) by entering the name of the command.

```
> edit(temp.cmd);
 {a screen is displayed}
= procedure MY_COMMAND is
=    A_VAR : FILE_TYPE;
=    begin
=      CREATE(A_VAR);
=      {more operations follow}
=    end MY_COMMAND;
 { the procedure is saved and
   edit is exitted }
> acli(input => temp.cmd;
>      spec  => temp.spec;
>      load  => temp.int);
: no errors were detected
: interface specification in "temp.spec"
: interpretable code      in "temp.int"
> my_command;
```

Figure 14: Sample User-Defined Command Procedure

In figure 14, the user enters the full screen editor (note that the unqualified name "edit" is sufficient, since a "USE screen_editor" statement is assumed to be contained in the system_env package). A command procedure is then created and saved. Next, the ACLI frontend is invoked to create an interpretable code module form of the command procedure and to create an interface entry for the command procedure. The user may then invoke the command procedure by simply entering the name of the command. The scope of the command is local to the user's session, since it was created under the session package. The user would have the option of promoting the command procedure definition to the permanent scope of the user's environment, by invoking an ACLI function.

1.5.4 Session Closure

At the end of a session with the system, the user ends the session by simply ending his currently active Ada proceudres. Entering "end USER_ID;" effectively completes the procedure body, and thus the user session. As a convenience, the ACLI will optionally confirm that the user intended to end the session. The user could choose to create a "logoff" command proceudres that would have the "end USER_ID;" appended as the last statement. Figure 15 illustrates both methods for ending a user session.

```
> end USER_ID;
: confirm session end
> OK;
```

Figure 15: Sample Session Closure

```
> logoff;
@ procedure LOGOFF is
@   begin
@     print(USER_LOG);
@   end LOGOFF;
@ end USER_ID;
```

Figure 16: Sample Session Closure with a Command Procedure

1.5.5 Special Features

1.5.5.1 Programmable Function Keys (PF Keys)

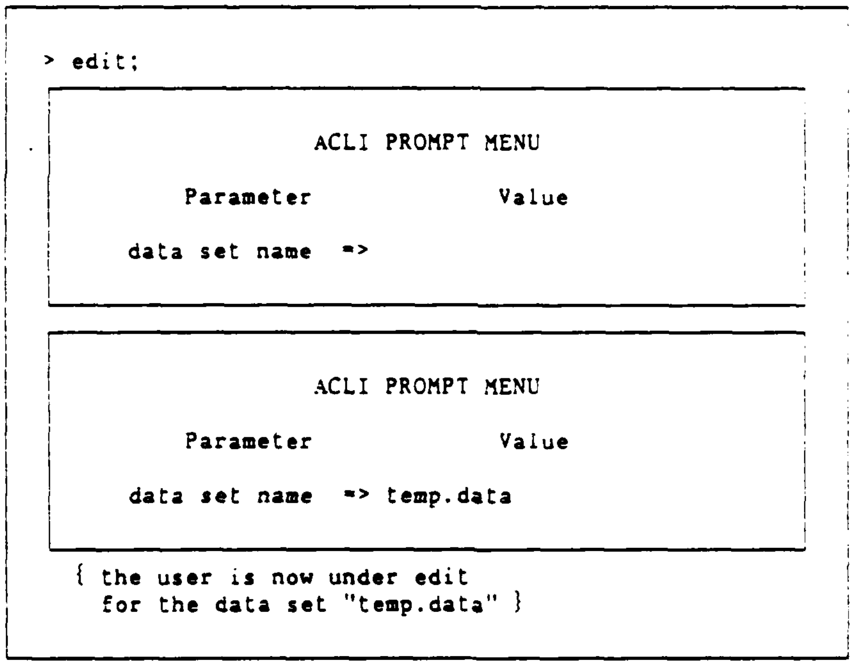
The ACLI accepts input from programmable function keys. The keys may represent text or imply a help or interrupt function.

1.5.5.2 Parameter Prompting

The ACLI will prompt for missing and/or incorrect parameters. Prompting is offered at several levels to accommodate experienced and novice users. Prompting may be explicitly changed on a command basis, allowing a user who is experienced with a set of commands to revert to a novice level for unfamiliar commands.

Prompting can be supplied on a full-screen or line-by-line basis, allowing terminals that are not VDT's to be used by the DCP. Programs that are invoked by the ACLI may issue prompts to the user, thus prompting may originate from both the ACLI and the functions it invokes. Prompting may be suppressed by changing the user's environment (suppressing prompting for all commands) or by requesting prompting/no-prompting on a command basis.

Prompting will be implemented as an Ada package, allowing the use of the prompting functions to be made available by any program that uses the prompt package.



```
> edit;

                    ACLI PROMPT MENU

          Parameter               Value

      data set name  =>

                    ACLI PROMPT MENU

          Parameter               Value

      data set name  => temp.data

   { the user is now under edit
     for the data set "temp.data" }
```

Figure 17: Sample Parameter Prompting by Menu

```
> edit;
: "edit" requires the parameter "dsname"
```

Figure 18: Sample Parameter Prompting without Menus

In figure 17, the user enters a command to invoke the editor. The editor requires a single parameter, causing the ACLI to display a standard full screen prompt with a list of the names of the parameters needed. The next screen shows the value, as entered by the user. After this screen, the editor is invoked by the ACLI with the single parameter.

Figure 18 shows the same action with and experienced user. The definition of experienced or novice user is retained in the user profile but may be changed on a command basis.

1.5.5.3 Help Services

The ACLI allows helps on much the same basis as prompting. Helps may be full screen or line-by-line. Helps may be suppressed on a user environment or command basis.

The ACLI organizes helps hierarchically, allowing a user to find progressively more detailed information. Helps may be traversed in a known fashion, including displaying a table of contents or index to a set of helps. Helps may be used to either explain the syntax/semantics of a command or as tutorials. Helps may also be related to components of a command. This allows help to be applied to a single parameter, or the parameter list, or any other command component that is useful.

The ACLI will display full screen helps for users that request the service. Help is requested either by invoking the "help" function or, more commonly, by pressing a help key. The help key may be pressed anytime and will be interpreted contextually by the ACLI. For example, pressing the help key after the "=>" symbol for assignment of a value to a parameter will result in the information about the parameter type, allowing the user to decide which value should be entered. Pressing the help key after the "(", at the start of a parameter list, will cause a help explaining the parameter list, with names and values. Help, like parameter prompting, may be tailored by the user on a command or permanent user environment basis.

```
> edit(
      {the help key is pressed}

                        ACLI HELP MENU

              The "edit" function takes a
          single required parameter and
          several, optional, parameters.

          PARAMETER         TYPE          ALLOWED
            NAME                           VALUES

          dsname          file_type
          mode            edit_mode     SCREEN/LINE

> dsname => temp.data);
  { the editor is invoked}
```

Figure 19: Sample Full Screen Help Menu

```
> edit(
      {the help key is pressed}
: enter the parameter "dsname" (file_type)
: and any of the following optional parameters:
:    mode (edit_mode)
> dsname => temp.data);
  { the editor is invoked}
```

Figure 20: Sample Line-by-Line Help Menu

In figure 19, the user enters a command to invoke the editor. The editor requires a single parameter. The user presses the help key after typing the opening parenthesis for the parameter list. The ACLI responds by presenting a help frame with a list of all parameters that the "edit" function accepts. The user may then refresh the screen and complete the command, entering the parameter by name.

1.5.6 System and Application Functions

The previous sections have explained the concepts behind command languages and interpreters and have presented several models for the Ada command language, interpreter, menu system and session. This section briefly presents some of the concepts implied in the command language model regarding the functions that the ACLI invokes.

In the ACLI model, each function referenced in a user command related to either:

- A system function that was directly invocable by the ACLI and required only an Ada specification to allow the ACLI to check the interface prior to directly calling the function.

- A load module that also had an Ada specification and allowed the ACLI to check the interface prior to invoking the loader to load and execute the function.

- A function that was internal to the ACLI and is, in effect, a subroutine. This type is the only type of DCP function that does not require an external Ada specification, since the specification is an integral part of the ACLI.

This section is concerned with the Ada specifications and bodies that must be created to allow the ACLI the ability to check a function's interface prior to executing the function (directly or indirectly).

The specifications will either relate to a body that is compiled Ada source, a load module that has been created in another language or a system function that is part of the host operating system. Specifications will be grouped in Ada packages, to express the relationship between the functions represented by the specifications.

Two main packages will exist: system and applications. The system package will contain all canned command procedures, commands and system functions that are an integral part of the ACLI subsystem. The application package will contain all other command procedures, commands and system functions that compose the DCP system.

The contents of the system package are listed in table 8, those already known for the application package are in table 9. Both of these lists are included in this concept paper to give an idea of the types of functions that may be expected to be invocable via the ACLI. Neither list is complete, and will in fact, form a dynamically alterable part of the DCP, as new functions are created and integrated into the DCP.

TABLE 8

Typical Functions in the System Package

TABLE 9

Typical Functions in the Application Package

1.6 GLOSSARY

ACL	The Ada Command Language. Ada, used as a command language interface between a user (or program) and the host operating system.
ACLI	The Ada Command Language Interpreter. The system-resident program that accepts commands in the Ada language and interprets them to invoke system functions, control sequencing of commands and other features of the ACL.
ACL System	The Ada Command Language Interpreter system consists of the major functions within the DCP that provide the interface between the host operating system and the user of the DCP. The ACL system is rehostable to a variety of computer systems.
Batch Mode	Batch mode is a method of having the command language interpreter execute a command without a user being interactively involved with the execution of the command.
Command	An Ada procedure. A command is invoked by simply entering the name of the command. A command is defined by precompiling Ada source through the ACLI frontend.
Command Package	A collection of Ada commands, types and variables, normally supporting a single type of object and operations upon it. A command package describes both a new type of object in the system and a set of operations for the object.
Command Procedure	The Ada procedure defining a sequence of actions, including invoking other command procedures, that may be invoked as a unit through entering the name of the command procedure.
Data Abstraction	The description of data in abstract terms. Describing data, without

	revealing the specifics about it, results in a conceptual understanding of the data. The specifics are handled by operations granted to the user of the data. Data abstraction allows localization of knowledge about data (and objects) and improves conceptualization of a system.
DCP System	The DCP system includes all of the software supporting the Distributed software engineering Control Process (DCP). The DCP system includes the ACL system, a database system, a library system and various other support systems and tools, all of which are packaged as the DCP system.
Environment	The user environment, also called the profile (c.f.).
Executable Form	The executable form of a system function allows the function to be activated in the system. An executable form is either a load module or an intermediate representation of ACL source, understandable by the ACLI.
Function	An invocable action within the sytem. A function is the same as an operation (c.f.).
Host System	The host system is the computer system that stores and executes the software of the DCP system. Many hosts may execute the DCP system, which is designed to be host-system independent.
Instantiation	The physical manifestation of a data abstraction. An instantiation allocates space within the system for an object of the type described in a data abstraction.
Interface	Information, understood by the ACLI, that describes a system function. The interface is expressed as an Ada specification for a command procedure or command package.

Object	An entity within the system. An object is understood by reading the data abstraction for the object and by examining the operations afforded for the object.
Operation	A system function that is invocable through the ACLI. An operation may be viewed as an action that may be applied to an object, or data abstraction.
Package	A collection of information (operations, types, variables, constants) relating to a single object.
Profile	User information kept between sessions. The profile is used to retain synonyms, parameter default values and other information that the user has control over and needs kept between sessions with the system.
Session	The dialogue between a user and the system. A session is defined as a sequence of commands and interactions with the system and its functions, beginning with a sign-on sequence and terminating with a sign-off sequence.
Specification	The Ada source that describes a command procedure in terms of name, parameters and types or a command package in similar terms.
Task	An activated operation within the system. A task may be a user session, a background job or a foreground job. Tasks are activated Ada procedures.
WITH	The Ada statement that allows all visible information within a package to be referenced within another package or procedure. WITH is used in the Ada command language to reference a package of commands, normally prefacing a user task. The WITH statement allows a command to be referenced, without it the command is undefined.

USE	The Ada statement that allows unqualified referencing to information made visible by a WITH statement (c.f.). The USE statement allows a command to be referenced without qualifying it with the name of the package.

LIST OF FIGURES

LIST OF TABLES

MILITARY MESSAGE PROCESSING

SECTION 1. OVERVIEW

The MAXI system was conceived and developed for the Air Force Intelligence Service (AFIS) Directorate of Intelligence Data Management (IND). The overall objective of the MAXI system is to provide a modular software suite to provide for the reliable reception and transmission of military message traffic as well as the applications necessary to disseminate, review, and coordinate messages within a local site. The MAXI system provides individual military sites of the intelligence community with a universal and standard set of procedures and software that is very reliable, and flexible in its configuration.

Early efforts at system integration were directed at establishing message preparation and transmission capabilities in the MAXI system. With these capabilities, a user can create and transmit messages to any destination serviced by the AUTODIN communications network.

Message dissemination to individuals, organizations, or functional divisions is performed through the Message Support System (MSS). MSS disseminates messages automatically without the need for human intervention. This is done with user-prepared message "profiles." The profiles are in essence, simple statements which tell the system that when certain words or phases appear in a message, it is to be routed to disseminees specified in the message profile.

MAXI now stands as the central component of AFIS's Common User's Baseline for the Intelligence Community (CUBIC). In this role, MAXI provides the CUBIC program and its users with the architectural framework and baseline message handling capabilities for a system that can accommodate current and future technological advances. MAXI may be used in the role of a stand-alone system, a communications link between systems, or as a front-end processor for other data processing systems.

MAXI currently provides message dissemination, message review, message preparation and transmission, message retrieval, temporary work storage facilities, terminal-to-terminal communications, report storage and generation, and interfaces to AMPE, LDMX, IDHSC II, and other external interfaces.

MAXI is the DODIIS standard AMHS and is currently operational at numerous sites both within the CONUS and overseas. Additionally, it is a prime candidate for early inclusion into the WIS software library.

SECTION 2. FUNCTIONAL REQUIREMENTS

The functional responsibility of the current MAXI software is to aid analysts and/or action officers in the efficient performance of their mission in a message communications environment. To this end, the software either performs or provides the user the ability to perform the following functions:

Command Entry

Text Editing

Data Queuing

Message Storage

Profiled Message Dissemination

Message Review

Message Retrieval

Work File Support

Message Generation, Coordination, and Transmission

Remote System Access

Analyst-to-Analyst Communications

Communications Support for other Applications Systems.

Each of these functional areas is discussed in the following paragraphs.

<u>2.1 Command Entry</u>. Operations available to the user are presented in an easy to understand format on display menus. The main branch menu lists all main functions of the system. Each main entry, when selected, presents one or more unique sub-branch menus depending on what subfunction is being performed. Also, any command requiring parameter

input, such as Work File interaction or printer selection, can be executed without parameters to receive a displayed list of the applicable values. All commands presented on the menus appear in both the three-character mnemonic format as well as being spelled out completely. The most often used commands are also selectable via function key. This feature is provided to reduce the number of input errors and to give the user positive identification of the available functions. Using the aids presented, the user may select one of three command entry mechanisms: light-pen (if available), typed-entry, or function key.

The typed-entry mechanism requires the user to actually type the command mnemonic on the command line and then execute it. When a function key is pressed the command line is bypassed altogether and the command is executed without delay. Selection of a command not valid to the current function results in a "bad command" error message being displayed. The terminal also provides a positive indication of when it is ready to process the next command, reducing the number of wasted keystrokes. The flexibility and positive feedback features of the user interface makes MAXI the most user-friendly system available.

2.2 Text Editing. The functions provided by MAXI for text manipulation are performed within the terminal microcode. These functions include:

Screen-to-Screen Move and Copy

Character-By-Character Insertion

Character and Word Deletion

Upper and Lower Case

Highlight

Automatic Reparagraphing

Screen Clearing

Cursor Positioning.

Two display modes are also provided: an unprotected free text mode allows data to be entered anywhere on the screen, while a protected forms mode allows data to be entered only in specified fields.

2.3 Data Queuing. In order to provide the user with the most flexibility in the performance of his functions, it is necessary to buffer him from the high rate of speed of data transfer. For example, if each message sent to a particular user were simply flashed to the screen and displayed there until the next message arrived, the user would be unable to review the messages fast enough to keep up. To this end a set of user queues has been designed to hold data until the user is ready to deal with it. Several of these queues are maintained for each user subarea defined in the system, each of which serves an independent functional purpose.

2.3.1 Message Review Queue. The message review queue is the holding queue for incoming message traffic. Messages on this queue are ordered chronologically by time of receipt within precedence (i.e., messages of the same precedence are grouped together with the highest precedence traffic first on the queue and the lowest precedence last).

2.3.2 INTRACOM Queue. The INTRACOM Queue is separate and unique from the Message Queue and is designed to hold analyst-to-analyst traffic in chronological order within one of the two precedence categories (Routine and Priority) associated with this type of traffic.

2.3.3 Work File Response Queue. The Work File Response Queue is designed to hold the Work File items retrieved in one retrieval operation. No precedence is assigned to such items, so the queue is structured simply on a first-in, first-out basis, determined by the order in which Work File items are found to satisfy the retrieval criteria specified in the retrieval query.

2.3.4 Pending Action Queue. The Pending Action Queue is a complex data structure designed to hold the responses to a message file retrieval query. Multiple query responses may be hold on the queue with each response having up to one hundred messages.

2.3.5 Hold Queue. The Hold Queue is a temporary holding area for work in progress. This queue is terminal, rather than subarea, related and is organized in a last-in, first-out sequence. This structure is analogous to a stack of papers on a desk, where the last item placed on the stack is the first item removed.

2.4 Message Storage. Incoming messages are stored in the master message file. This file is organized into physical segments or "day files" for efficient use of disk space and for convenient reallocation of the oldest used space for new traffic (the day file purge process). The configuration of the message file may be tailored at each site, the requirement being to provide a minimum of thirty days of traffic stored on line regardless of daily traffic volume or average message size. Each message on the file is stored with associated records containing information concerning which subareas received a copy of the message and what subject codes were assigned to it in the dissemination process.

2.5 Profiled Dissemination. The Profiled Dissemination function provides a mechanism by which each user specifies what subset of the incoming message traffic he desires to receive. The user-generated "profiles" are compiled into a machine-readable "dictionary" which controls the dissemination process. Incoming messages are scanned for the presence of any of the user-specified keywords or phrases. Once all key items have been located, a compare is performed against the logical relationships defined in each profile. A match results in dissemination of the message to each of the subarea queues, and assignment of each subject code specified on the matching profile or profiles.

2.6 Message Review. The Message Review function provides the user with the ability to view message traffic queued to his station. Messages on this queue are organized in chronological order of receipt within precedence with FLASH or CRITIC messages causing an audible and visible alarm to be presented to the user. The queue can be easily manipulated in a forward or backward step fashion as well as by large positional jumps. Messages can be deleted from the queue individually or in groups. Group deletions, or purges, remove a user-specified number of oldest messages on the queue regardless of precedence. The system also provides for temporary distractions, allowing the user to leave the review function and return to it later without losing his place. Messages on the queue may be rerouted to other subarea queues and can have additional subject codes assigned for use in later retrievals. Messages displayed for review may be modified and/or stored or printed for personal use, but it should be noted that the message file copy will remain in its original form; no modifications to this copy being allowed.

2.7 Message Retrieval. The Message Retrieval function provides the user with the capability to retrieve a single message or a group of messages from the master message file, by using a single parameter or a combination of retrieval parameters. These parameters include the message's originator, date-time group (DTG), Station Serial Number (SSN), and any subject index terms or disseminees derived from the profile keywords. Specific messages can be retrieved via DTG only, DTG and originator, DTG and SSN, or DTG, originator, and SSN. Groups of messages may be retrieved by a range of DTGs in combination with originator and/or subject index codes and/or disseminee codes. Subject index terms can be specified in logical (AND, OR, BUT NOT) relationships with other subject index terms or disseminee codes to select the exact subset of messages desired. Naturally, the more specific the search parameters specified, the smaller the number of message satisfying the criteria. Retrievals are validated prior to execution to prevent time-consuming searches for non-existent subject or disseminee terms.

2.8 Work File Support. The Work File capability provides the user with an indexed storage area in which he can save information of interest. One Work File is provided for each subarea defined in the system. When storing a display item in a Work File, one or more user-defined titles can be assigned to the item. In this case the item may be later retrieved using either title. Alternatively, multiple items may be stored under the same title. Additionally, each item is assigned a storage date-time group. A single date-time group or a range of date-time groups may be specified as retrieval parameters, in which case all items whose storage times fall into the specified range or match the single specified date-time group will be returned to the user's Response Queue. Currently, the Work File Subsystem is capable of storing up to 2048 items under 256 different index titles, allowing an average of eight items per title. Work Files are also "protected" to three levels:

- o Read only
- o Read/Write
- o No Access.

These privileges are established by the system manager and may be revised at any time. The Work File subsystem is also provided with a full backup and recovery capability.

<u>2.9 Message Generation and Transmission</u>. The message generation and transmission function allows the system user to generate a message for transmission with a minimum of knowledge of specific network protocols. Using easily completed forms the user first builds the message header, assigning precedence and security values, then completes the addressee portion of the message, providing Routing Indicators and/or Plain Language Addresses (PLA) or Address Indicator Groups (AIG). Finally, he completes the text of the message in free text mode. This process provides the greatest flexibility, allowing use of previously completed forms for each phase of mesage construction. Use of free text mode for message text generation allows full use of the text editing features of the terminal. Once the message has been generated, a mechanism has been

provided to prevent the originator from transmitting the message without the explicit approval of a "releasing authority", an individual assigned to determine that the message is of the proper security classification and that it is otherwise ready for release.

On approval, the message is transferred electronically to the network interface software for transmission, where it is converted to line format according to the protocol requirements of the network.

<u>2.10 Remote System Access</u>. Remote System Access is a general capability designed to provide the user with access to other remotely located systems. Standard MAXI terminal support software is used to connect to a local (resident in the MAXI configuration) service program. A communications link is established, providing the user with a terminal interface to the selected system as if he were locally connected to it. This capability provides for remote analyst-to-analyst communications and ancillary data base access.

<u>2.11 Performance Characteristics</u>. Overall system performance characteristics covers the availability, reliability, responsiveness and throughput rates as they concern the functional operation of the system.

Response time is defined as the time between a specific stimulus and the completion of the associated required response. Response times currently mandated and being accomplished by the MAXI system include:

o	Queue commands	4 seconds
o	Message commands	1 second
o	Terminal commands	1 second
o	Analyst-to-Analyst	2 seconds

Message processing speed is dependent upon the specific priority of the given message being processed. Messages of lower priority have their processing suspended when a message of higher priority is received. The time required to process FLASH messages received from external systems to the point where they are ready to display during times of peak system loading does not exceed 15 seconds.

Throughput rate is defined as the average throughput obtainable during time of peak system loading. When utilizing automatic input channels (AMPE, CSP, etc.) MAXI currently meets the original design requirement of 500 AUTODIN transmitted (2000-40,000 characters) messages per hour. In a peak system loading condition, MAXI frequently exceeds 200 outgoing AUTODIN messages per hour. Current operational message volumes supported vary between 800-2500 messages per day.

SECTION 3. CASE STUDY

NAME OF SYSTEM:	Modular Architecture for the Exchange of Intelligence (MAXI)
SPONSORING AGENCY:	Air Force Intelligence Service (AFIS), under an Executive Agreement with the Defense Intelligence Agency (DIA)
DEVELOPMENT CONTRACTOR:	INCO, Incorporated C^3I/AMHS Department 8260 Greensboro Drive McLean, Virginia 22102 Attn: Mr. Dick May
DESCRIPTION OF SYSTEM:	A generic overview of the system is provided in Section 2, Functional Requirements. MAXI has been mandated by DIA as the Department of Defense Intelligence Information System (DODIIS) standard automated message processing system. As such it serves as the foundation for the AFIS-managed Common User Baseline for the Intelligence Community (CUBIC). Additional information regarding system attributes and performance may be obtained by contacting AFIS directly.
QUALITY OF SYSTEM:	MAXI is currently an operational system at 14 intelligence sites and 2 command and control sites throughout the world.
DEVELOPMENT ISSUES:	The current MAXI system is the culmination of years of experience in the development of automated message processing systems. MAXI has its roots in the original National Military Intelligence Center (NMIC) message processing system; as well as the PACOM Data Handling System (PDSC) originally developed for PACOM Headquarters. The MAXI system originated as a melding of the most promising features of these two preceding systems and has been designed and developed as a modular software system in order to adapt to the technical evolution required by today's command and control environment. MAXI currently utilizes MACRO-11 as the source language of choice and resides on the AN/GYQ-21(V) as the intelligence community hardware architecture suite.

END USE:	MAXI was initially fielded at Headquarters, Military Airlift Command in March of 1981. Currently the MAXI system is in operational use at the following sites:

1. HQ MAC, Scott AFB, ILL, MAR 81
2. HQ ESC, Kelly AFB, TX, JAN 82
3. HQ USAFE, Ramstein AFB, GE, AUG 81
4. 497th RTC, Shierstein, GE, FEB 83
5. TAWC, Eglin AFB, FL, FEB 82
6. AFAITC, Lowry AFB, CO, JUL 81
7. HQ TAC, Langley AFB, VA, DEC 82
8. HQ ADCOM, NCMC, Colo. Springs, CO, JUN 83
9. HQ PACAF, Hickam AFB, HI, JUN 83
10. HQ REDCOM/J2, MacDill AFB, FL, MAY 83
11. HQ REDCOM/J3, MacDill AFB, FL, AUG 83
12. FICEURLANT, Norfolk, VA, JUL 83
13. HQ USAREUR, Heidelberg, GE, JAN 83
14. HQ USAFE/OSC, Ramstein AFB, GE, SEP 82
15. HQ USAFE/TFC, Boerfink, GE, JUN 83
16. AFIS/IND, Bolling AFB, D.C., ARP 83

New MAXI installations expected to be accomplished during FY 1984 include:

HQ USSOUTHCOM, Panama

KISS, Korea

Alaskan Air Command, Alaska

USCENTCOM, MacDill AFB, FL

SHAPE Headquarters, Belgium

Army Operations Center, Pentagon

TCATA, Ft. Hood, TX

SECTION 4. ANALYSIS

The cost of this conversion effort has been derived using a structured software life-cycle analysis. The overall approach to this analysis relies on the following assumptions:

- o Conversion to Ada is essentially a development task
- o Ada compilers and language systems will be available and provided for the effort.
- o No major functional redesign of the current MAXI capabilities is required.

The software life-cycle analysis comprises six subtasks that are organized and performed linearly but incorporate necessary feed-back loops to provide the requisite checks on quality. These subtasks, as depicted in Figure 4-1, are:

- o System Requirements Definition
- o System Architecture Definition
- o Detailed Design
- o Code/Unit Test
- o Integration Testing
- o System Testing - Acceptance

Since the MAXI functions and requirements are well-known and embodied in an operational system, this subtask is unnecessary in this effort. However, Ada mandates a software architectural redefinition for these requirements. Furthermore a cadre of trained Ada software engineers is required to perform the conversion.

Figure 4-2 presents a GANTT chart that lays out these subtasks and their milestones. On the chart, the system architecture and detailed

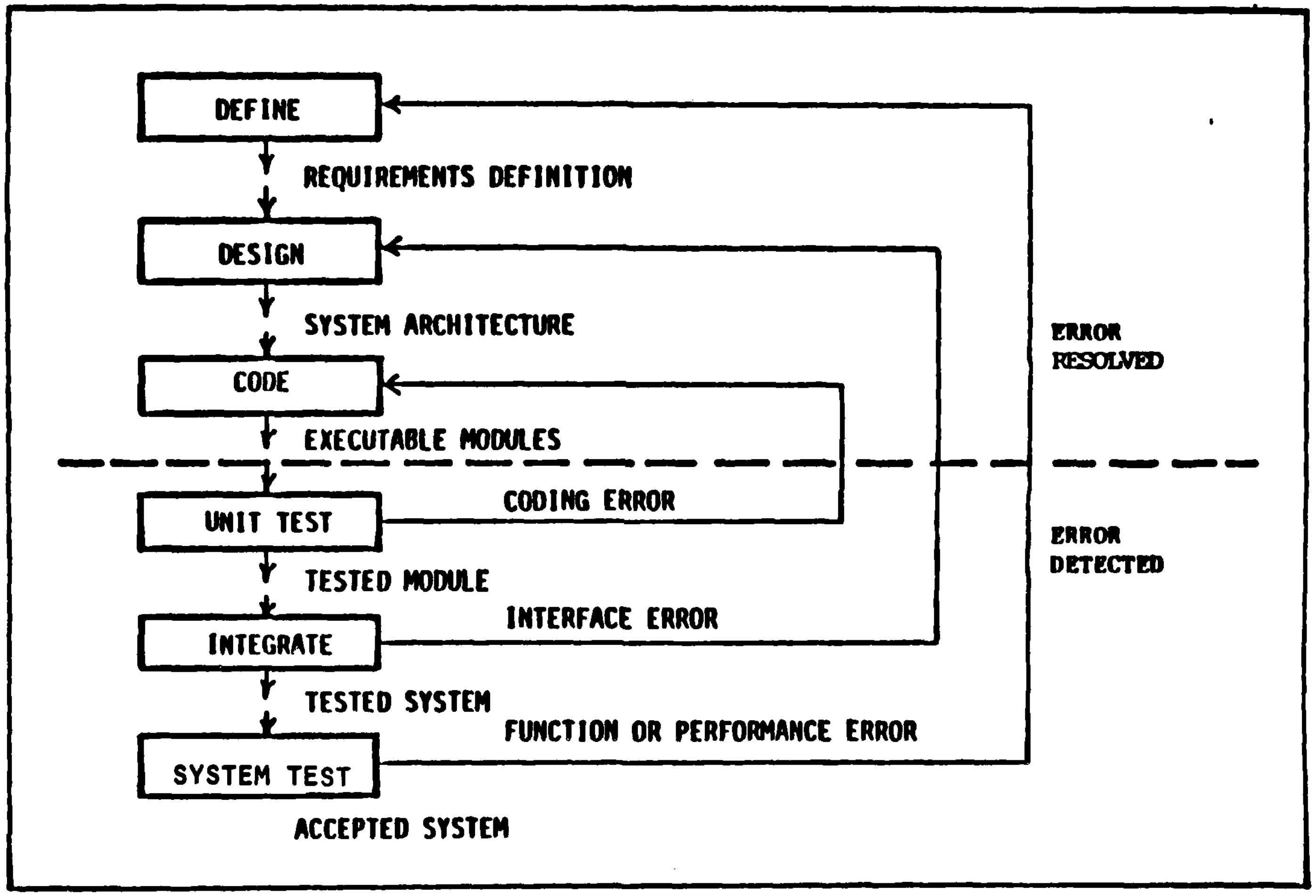


FIGURE 4-1. DEVELOPMENT SUB-TASKS

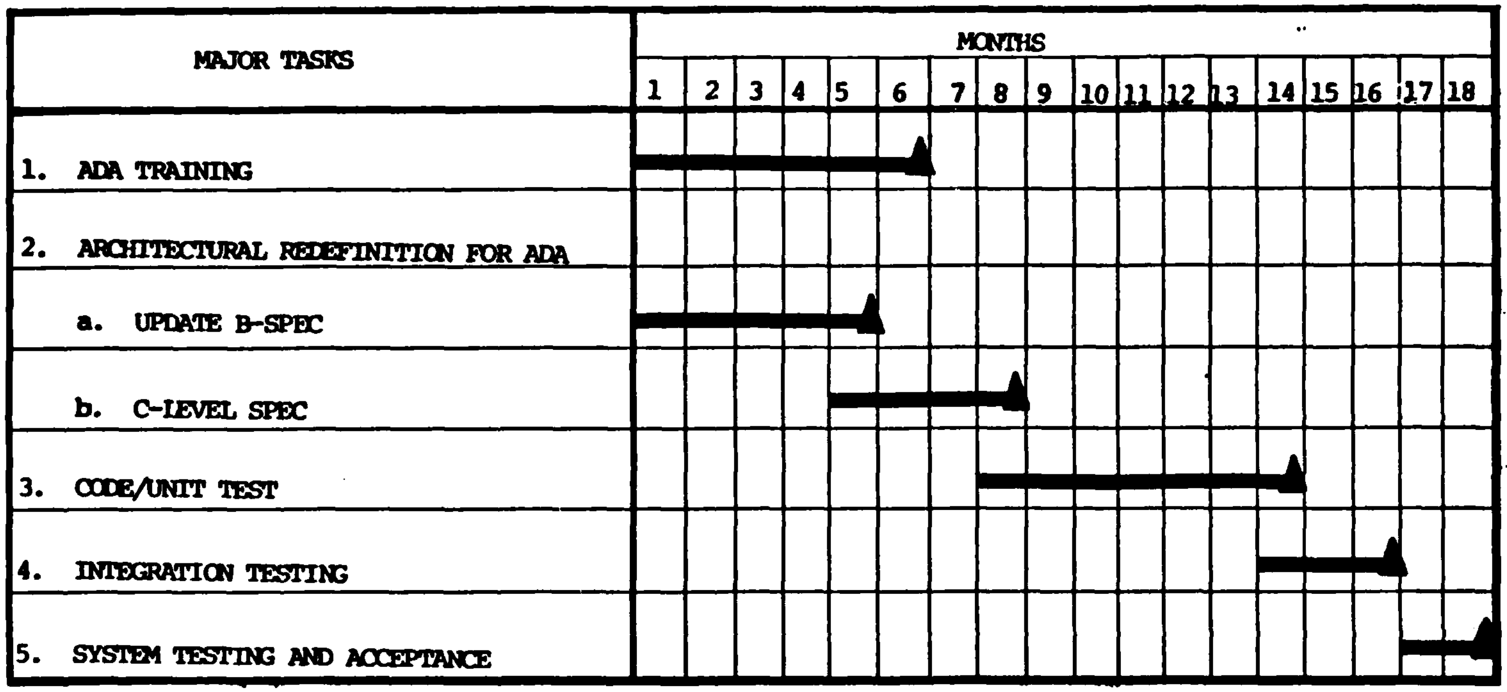


Figure 4-2. Subtask Milestones

design subtasks have been combined under a common heading. A relatively small core of software engineers will spearhead the architecture conversion while the remainder of the development team is rigorously trained in Ada. This core will then head the various component developments identified by the Ada architecture. Implicit in the estimates are the support personnel needed for documentation, configuration mangement, and quality assurance testing. The man-months estimates by subtask are the following:

SUBTASK	ESTIMATE IN MAN-MONTHS
Ada Training	72
Architecture Redefinition	25
Detailed Design	96
Code/Unit Test	119
Integration Testing	72
System Testing/Acceptance	48

SECTION 5. CONCLUSIONS

The MAXI automated message handling system is the culmination of more than seven years of research, development, implementation, and enhancement. A direct result of the transfer of technology from the NMIC and PDSC systems, MAXI is currently operational at sixteen sites both within the CONUS and overseas. It is the standard single processor AMHS for DoD's Common Users Baseline for the Intelligence Community, and is a prime candidate for early inclusion in the WIS/CUS software library.

COMMAND INFORMATION MANAGEMENT:

THE COMMANDER'S WORKSTATION*

Daniel J. Power

* I want to thank Carol Pokodner for her help, ideas and encourgement. Also, I want to thank the people at Apple, Masscomp, Mesa Technology and Xerox, especially Eva McGhee and Frank Shap, for their assistance and ideas. John Sapp, Software A&E, has encouraged me and helped me meet my deadline.

September 16, 1983

October 28, 1983

Dr. Tom Probert
Institute for Defense Analysis
1801 N. Beauregard St.
Alexandria, VA 22311

Dear Dr. Probert:

The Institute for Defense Analysis (IDA) has my permission to reproduce my paper "Information Management: The Commander's Workstation" for any uses in conjunction with the WIS project.

Sincerely,

[illegible]

Daniel J. Power

OVERVIEW

What is a Commander's Workstation (CWS)? CWS is envisioned as an integrated hardware/software computer product that provides military officers with powerful capabilities for performing their jobs. CWS should be a stand-alone machine that has easy-to-use, pointer-controlled software, a large memory, high resolution graphics, and sophisticated voice and data links to other CWS, external data bases, and analog devices. The integrated CWS software command environment should provide the following general capabilities: screen-oriented document processing, data base query, automatic message routing, a schedule and reminder system, access to maps, books, papers, graphics and map processing, automatic data capture and transfer to update data bases, logistics management and ordering, and encryption/decryption capability for voice and data. Specialized capabilities should also be available in CWS to support those commanders that require them including: planning tools, planning checklists, project management, simulations, optimization programs for military mission assignment, and military unit readiness, personnel and status monitoring.

A Commander's Workstation can be produced with current and expected short-term enhancements to hardware and software technology. Apple Computer, Data General, Masscomp and Xerox, in particular, have systems in various stages of development and distribution that can provide many of the needed hardware and software capabilities. Peripheral manufacturers and third-party vendors can supply additional hardware and software requirements. Developing a field version of CWS may require a sizeable investment, but vendors are developing many portable machines for business executives.

Developing and testing CWS will require more than 150 person-years of effort and will cost more than $16 million dollars (in current 1983 dollars), but CWS development costs are minimal when compared to the cost of implementing the concept throughout DoD. Some may argue that ultimately only a few thousand machines will be needed, but I agree with Druzhinin and Kontorov (1972), Soviet military technology experts, that CWS must be universally distributed in the military chain-of-command to realize the following benefits: improved operational capabilities; faster and more accurate planning for strategy and tactics; and reliable and effective command, communication and control.

If military and political leaders are willing to make a strong commitment to the project and if adequate resources are provided, it is in my opinion that it is feasible to develop CWS by April 1987 and 60,000 to 100,000 CWS can be operational by January 1, 1989.

INTRODUCTION

This paper explains and evaluates a product called a Commander's Workstation (CWS). First, two scenarios about the product as it will be used are developed. Second, technical specifications for the product are provided. Third, capabilities of currently operational hardware and software are reviewed. Fourth, the potential availability of key product capabilities during the next five years is briefly discusssed. Finally, costs, schedules, and a possible development plan are summarized. The conclusion of the paper is that WWMCCS can achieve significant improvements in defense planning; improved logistics support; and perhaps most importantly more effective command, communication and control capabilities if a commander's workstation is developed and implemented throughout the military chain-of-command.

SCENARIOS OF CWS IN USE

Two scenarios are presented below that describe the use of a Commander's Workstation in 1989. I developed the first scenario and the second is based by Druzhinin and Kontorov (1972), Soviet military technology experts. Chapter 14 of Druzhinin and Kontorov's book is included as Appendix I. Chapter 14 contains the scenario and proposes three expert systems that may be feasible to implement on the workstation by 1989.

Scenario I

General Smith arrives at his office at 0730. He walks to his command workstation and places his thumb on the identification recognition unit. The recognition unit moves the system from standby to a full activation status. The speech synthesis unit, used in the workstation, responds, "Good morning, General Smith." General Smith glances at the terminal display and notes that all of his waiting messages are status two or three, which are lower level priority messages. He also notes the operations monitoring code of the system is green indicating the troops under his command meet general readiness requirements and no major deviations have been detected from his readiness standards.

After having a cup of coffee, General Smith sits down at the workstation and uses his hand controller to indicate he wants the main menu. The menu is displayed on the screen and Smith selects the news headline file.

The news headline file contains information about major world, national and military events that have occurred since he last scanned the file. The information is automatically stored and updated at his workstation. Smith quickly scans the headline briefing file and notes certain information he wants to store for possible later reference. To save a briefing he positions his hand controller at the reference number and pushes the controller button to store the information in an indexed storage file.

After reviewing the headline briefings, Smith returns to the main menu and uses the hand controller to bring up the current messages file. He reviews messages that have been received overnight. As he encounters a message he feels needs a response, he moves his controller to the top of the screen and activates the "message response" block. He pushes a button and overlaid on the screen is a separate work area that contains a memo form. Smith uses his right hand and the executive keyboard to indicate the name of the person to receive the response and the message. He moves the hand control pointer to the top of the screen, to the "send message" box, and pushes the button on his hand controller, to automatically route the message to the person. The message is stored and cross-referenced with the original message. Smith normally keeps records of his messages in active storage for two weeks and the system automatically moves those messages to permanent long-term storage to maintain an archive of his command activities.

After reviewing his messages and sending out responses where needed, Smith returns to the main menu and pushes the "status" menu selection. Displayed on the screen is the status report of the units under his direct command. Smith can move the pointer, using the hand controller, to a specific military unit, and request additional status information about that unit. As he reviews the status of units under his command, Smith is able to again send messages either telemail messages or voice messages, using the built-in telecommunications system at his workstation. Smith usually requests written reports from subordinate commanders, but sometimes speaks with them personally.

Following the status review, Smith notes that the status light at the bottom of his screen is flashing red, indicating an important message is about to interrupt his use of the terminal. Almost immediately after the status light flashed, the terminal cleared and a message was displayed indicating he was urgently needed at a command briefing. Smith returned to the main menu and moved his pointer to the "standby position" command and the workstation changed mode. Smith left to attend the command briefing.

When he returned, he again used his thumb print as an identifier and again noted no important messages were waiting and the status of units was green.

Smith called up the main menu and requested the planning program. Smith had been contemplating a war games exercise for units in his command. He decided the project management program should be used to prepare activities and a plan for the military exercises. Smith also used a modelling program to attempt to simulate possible outcomes of the exercises, given the current status of units under his command. This simulation enabled him to assign units to opposing sides to ensure that each side was about equal in capabilities. After preparing the activities, assignments, time schedule, running the simulation and indicating which units would be on the two opposing sides, he prepared a directive. The directive included an activities list with responsibilities and time deadlines and assignments to the red and blue teams for the exercise.

Smith reviewed this document in the "editing" mode, added comments in appropriate places and determined that the information should be reviewed by three of his subordinates. He moved the hand controller to the top of the screen and activated the "message transmittal" mode and a second screen appeared in the lower right hand corner. Smith indicated the three subordinates that were to receive his message and then typed "current document" as the contents of the message. Smith moved his pointer to the "transmit box" at the top of the screen, pushed the button on the hand controller and the message was automatically sent to the three subordinates. Smith quickly moved the pointer to "save", and the plans for the military exercise were stored so that he could review and update them as needed.

Smith decided he could best spend his time after lunch reviewing some military periodicals and returned to the main menu to activate the data retrieval mode. He typed in the topics he was interested in and indicated he wanted articles that appeared in the previous two months. As the search was occurring Smith returned to the main menu and used the "logistics support" module to order three terrain maps of his command region and a new uniform from the central supply office. When the terminal display indicated, in the lower left hand corner, that his search was complete, Smith returned to the main menu. He positioned the pointer over "data retrieval" and the screen had a list of articles in the topic areas he requested. Smith moved the pointer over the articles, scanning the titles. He noted one title which appeared especially interesting and when he pushed the button on his hand controller, the article appeared on his screen. He scanned the abstract and decided the article seemed quite appropriate to his needs. Smith read the article on the screen and decided after reading it that he wanted to keep it in his files for easy reference during a meeting with subordinates later in the week. He moved the pointer to the top of the screen, pressed the control at the "save" block, and then moved the pointer to the "remind" block. A window appeared on the screen and the system queried "WHEN DO YOU WANT TO BE REMINDED ABOUT THIS DOCUMENT?" Smith typed in 1400, Friday, 9/14/83. The system responded

"REMINDER NOTED" and the window closed. Smith returned to the article list and scanned a few other articles, saving some, rejecting most of them. When finished, he returned to the main menu.

Near the end of the work day, Smith requested the "message" mode and sent a message to two subordinates located at his headquarters unit requesting that they come to his office in fifteen minutes for a briefing. The subordinates acknowledged receipt of the memos. While waiting for his subordinates, he worked on routine matters at the workstation. He reviewed and corrected a draft of a document he had dictated at his workstation that his secretary entered and transmitted back to his workstation.

Fifteen minutes later, the subordinates knocked on his door and he greeted them. They entered and Smith pushed a button on his console that switched the display from the workstation monitor to a large-screen monitor on the wall in his office. Smith walked over to one of the five chairs around the screen and sat down and took the hand controller in his left hand. A keyboard was attached to a movable arm at his right hand. Smith explained the purpose of the briefing and retrieved the planning documents he saved earlier and displayed them on the screen. He and his subordinates discussed his plans for the upcoming military exercises. As they discussed them, Smith entered the changes in the display. They retrieved maps of the terrain from laser disk storage and noted the potential effects on the exercise. After 45 minutes of reviewing the plans and terrain for the military exercises, Smith adjourned the meeting, saved the new plans, and routed copies of the plans to each subordinate. Smith then returned to the main menu and selected the "standby" icon closing the workstation down and went home after a long day at the office.

Scenario II

Major Jones arrives at a new assignment and, after becoming familiar with his deputies and closest associates, he sits at the Commander's Workstation to conduct a situation analysis. Following thumb print identification, he uses his hand controller to select maps displaying grouping of forces in the immediate area of combat operations, first in small scale: the enemy, our forces, and neighbors; and then changing to a larger scale and other displays; the enemy disposition of troops and fighting strength finally zooming in on enemy strong points and weapons placements. He next requests with the pointer quantitative data on the makeup of enemy units, characteristics of their military equipment, military experience, and morale of their troops, as well as lists and combat characteristics of the command staff. He prints any information needed. If there are no data on certain questions, these questions are directed using the "memo" mode to the intelligence branch for the collection of additional information.

Major Jones then undertakes initial familiarization with his own troops. Using the workstation and interaction with his closest aids, he becomes familiar with the disposition, makeup of troops, weapons, equipment, data on combat capability, combat readiness, morale of personnel, and the combat and psychological characteristics of commanders directly subordinate to him. Again using the "memo" mode for unanswered questions, he requests information from subordinates and staff sections. Descriptions of present and future missions are displayed. Then the commander may review previous events. If necessary, the course of combat actions in a given region can be reconstructed on the screen (in convenient time scales). Proposed, but unimplemented decisions are noted. This phase of interaction with the work station ends in familiarization with current combat orders and instructions, logistics, updates and contact with subordinate commanders of various ranks and neighboring units (automatic communication and data display are used here).

The next step is to analyze the theater of military operations (terrain). Photographs of various battle areas and the surrounding sectors are displayed on the screen. By changing the scale using the hand controller, the commander can acquire the most informative representation of the data, which are simultaneously supplemented with digital data and text. The commander and his aides analyze the terrain, weather situation, and geophysical conditions, using an interactive query system. The subordinates, in turn, direct the commander's attention to details which they think are important. The mission assigned to the troops is shown on large-scale stereoprojections of the terrain.

This phase is concluded by using simulation and planning programs to consider possible variations of combat actions, logistics, preparatory measures and their organization. The purpose is to concentrate on those questions which arose in the commander's mind as a result of his initial impression. Some of the tasks supported by the work station are specific in character: prepare mathematical data, evaluate the effectiveness of certain measures and actions. Other tasks are more general: assist in proposing solutions for problems and subproblems, assist in forecasting the situation, assist in evaluating the enemy's intentions. An expert program helps the commander with questions such as: "What is the probability that the enemy will attack at sector A within the next 2 days?" The system may answer approximately as follows: "For the next 2 days the enemy may concentrate such and such forces in sector A; the effectiveness of attack is such and such, the probability of attack is such and suc'i."

Having formulated the questions and designated the time frame that he wants for answers, the commander may visit his troops for on the spot familiarization with the situation. In this stage of the operation the work stations of subordinate commanders can be used.

In the next step, the major returns to his command post with new ideas and impressions, and begins to make decisions. But first he listens to subordinates and discusses the answers from the expert system. He uses the workstation to collect additional information relative to the situation, state of enemy troops and his own troops. Then he formulates his initial ideas on military tactics. The major transmits his proposal to headquarters, including overall objective of the forthcoming battle. At his workstation, he can recruit supervisory staff replacements from the field. He makes suggestions that are fed into the computer for evaluation and transmission to headquarters.

The next step may be characterized as the discussion stage between the major, staff and support services using encrypted voice communication. Using the document processor, reports are prepared; an expert system evaluates the proposals made during the "brainstorming," analyzes them, separates the constructive ideas and uses them for suggesting alternatives. Key personnel formulate their own ideas and proposal, and feed them into the expert system for combined analysis and aggregation.

Finally, military tactics are authorized at appropriate levels and command directives are automatically sent to workstations.

Other Scenarios

These two scenarios only begin to describe the uses and applications of CWS. Scenarios for all of the services, all command levels and major anticipated events, e.g., a new assignment, routine management, should be constructed. Scenerios help evalute the capabilities of the CWS before large investments have been made in design and development. Also, scenarios can provide guidance to designers.

FUNCTIONAL REQUIREMENTS FOR COMMANDER'S WORKSTATION

The Commander's Workstation (CWS) must have the power and flexibility to serve military commanders at all levels in the command structure and it must have capabilities for supporting planning, logistics, and command, communication and control in four types of situations. The four situations include:

1. Routine administrative and operational management
2. Crisis management and planning
3. Conflict situation management and planning for:
    a. limited conventional warfare
    b. global conventional warfare
    c. limited nuclear warfare
    d. global nuclear warfare
4. Post global nuclear warfare management and planning

The commander's workstation must have software capabilities to support the following generic management tasks:

1. military planning, both strategic and tactical
2. organizing military units
3. delegating tasks and responsibilities
4. monitoring and directing the actions of subordinates
5. staffing military units, including planning, readiness, personnel transfers, skills inventories
6. searching for information
7. routine decision making

Both the hardware and software for CWS must be carefully chosen to ensure modularity, expandibility, user friendliness, and reliability.

Required Hardware

The following hardware and software requirements seem necessary to provide the capabilities in the scenarios and meet requirements in the four military situations for supporting the seven management tasks:

1. a machine with a minimum of a 32 bit main microprocessor; separate CPUs for screen output, multiple input channels, and memory management; clock, a minimum of 1.5 megabytes of RAM, 500K ROM; printer and communications buffers; with expansion slots; and easy upgrade of components to incorporate new hardware and software capabilities.

2. thumb-print recognition unit
3. a 35 megabyte hard disk
4. an easy-to-use keyboard for non-typists
5. built-in speaker phone and speech recording
6. a "mouse" or pointer control device
7. 25" diagonal display with high resolution bitmap capability and fast refresh rate, 132 column display capability
8. 160 cps printer with graphics
9. modem
10. back-up power supply

Optional Hardware

In some command offices, enhanced capabilities should be available. The following capabilities facilitate group meetings, information search, field command, and intelligence:

1. read/write laser disc storage
2. 5 ft. projection of flat display screen
3. voice synthesis and recognition
4. bubble memory storage
5. compact field unit with bubble memory, flexible membrane keyboard and flat screen display
6. 40 ips plotter
7. video imaging input camera
8. digital/analog converters
9. laser printer
10. portable power supply
11. double-sided, quad-density, floppy disk drive (should be in limited use to main information security)

Software

A complete specification of software for specific command situations is not possible at this time, but at a minimum the following is needed:

1. an integrated software environment manager, captures all data descriptors and is transparent to the user, uses graphics symbols with mouse or keyboard control

2. sophisticated office management software, e.g. screen-oriented document processing, communications, file management, schedule and reminder system, data base query system, spelling checker

3. graphics software management system, with retrieve, modify, rotate, create, combine with text, zoom, roam

4. planning tools, planning checklists, project management, simulation of military situations, optimization of air craft, ships, military units (cf., Pazzani, 1983), statistical analysis, budgeting

5. voice message management and storage

6. automatic data capture and transfer software

7. military readiness, personnel and status monitoring software, with user control of readiness measures and standards for units and capabilities

8. logistics management and ordering systems

9. encryption/decryption software for voice and data

CAPABILITIES OF OPERATIONAL SYSTEMS

In this section, operational workstations of four manufactures are brielfy reviewed. The four manufactures are: Apple, Data General, Masscomp, and Xerox.

Apple Computer has an advanced workstation called Lisa priced at $10,000. This product does not currently meet the hardware specifications for CWS, but the software environment is state-of-the-art. Lisa is very easy to use and novices can learn the system quickly. Lisa is an integrated software environment with advanced office automation tools. The graphics are excellent, but higher pixel density is needed. The system can be, and most certainly will be, expanded and enhanced. Lisa has seven major software components that could be used in CWS, including a project management program.

Software development tools will be available for Lisa. Pascal, Cobol and Fortran are currently available and other languages should be available in 6-9 months. Capabilities for developing expert systems are not currently available.

Data General is introducing an advanced workstation, GW/4000, priced at $80,000. GW/4000 is a more powerful machine than Lisa, but it does not currently meet all hardware requirements. Word processing and communications are currently available, but the machine seems intended for data processing. I have not used this machine nor seen it in operation, but the product specifications suggest Data Geňeral may be able to bid on this project.

Masscomp is a small, entrepreneurial company that is introducing an advanced workstation priced at $14,000. On paper this machine comes closest to the hardware requirements for CWS. The machine has a 10MHZ 32-bit VLSI processor with a 4K byte cache. The system uses a UNIX operating system and has high-resolution graphics. The current software is limited, but icons and a mouse are used. I have spoken with sales representatives of Masscomp and seen the Graphics Cluster workstation. This product definitely has potential and further evaluation is needed.

Xerox is the orginator of advanced workstations that use icons. Two products are potential starting points for the CWS, the Xerox 8010 "Star", and Xerox 1108. These two machines are very powerful and sophisticated. The 8010 is priced at about $14,000 and meets the reqirements for an advanced office automation environment, including software. The 1108, priced at about $26,000 is an advanced artificial intelligence (AI) machine. A machine that combines the capabilities of these two machines (with technological updates) may meet the requirements for CWS better than any other system reviewed. Xerox is in a position to provide many powerful AI and software development tools if they aid this project to develop CWS.

Systems produced by PERQ, Teletype Corporation Hewlett-Packard (H-P 9000) and Santa Barbara Development Labs should also be reviewed and evaluated.

ANALYSIS OF SUBSYSTEMS

1. Large screens. The projection TV technology is improving and many vendors have products. The projection systems are still large and bulky, but miniaturization is occuring. *Flat screens may* fill the need for large display screens. Costs will run from $2000-$20,000.

2. Laser disk storage. Matsushita Electrical Industrial Co., Ltd. has announced an erasable optical disk. The disk has a capacity of about 1,000 Mbytes. The systems should be available in 1985 for about $35,000.

3. Voice synthesis and recognition. Software and hardware advances are occuring rapidly. Texas Intrustrments will soon be releasing an advanced system.

4. Bubble memory storage. The technology is available for field units and it will be improved despite the exit of some vendors from this product area.

5. Field unit. Engineering work and technological developments are needed, but a unit can be available in 1986-87. A separate project will be needed to develop this through an OEM, etc.,

6. Plotter. Many plotters are currently available and speeds will increase.

7. Video imaging input camera. Some large input centers will be needed to create laser disks and route images to workstations. Also, some hand-held units will be needed. A number of systems are currently available, including systems from Digithurst, Micron Technology and AUDRE, Inc.

8. Laser printer. Xerox and Quality Micro Systems and other vendors have laser printers. Speeds should be in excess of 120 pages per minute by 1985.

9. 25" diagonal display. TEK has a 25" diagonal display with 4096 x 1096 points addressable, refresh 537 m/s. These displays should fall in price in the next 2 years.

10. Thumb-print recognition unit. NSA will probably need to do R&D to create this product.

11. Keyboard for non-typists. Prototypes need to be built for field testing. Cost for development and testing of approximately $500,000.

12. Software. Both the Apple and Xerox systems have important software components for CWS. Data base management systems can provide a starting point for other capabilities. A sophisticated software development environment will be necessary to keep costs down and both Apple and Xerox are able to make these environments available for this type of project. On the Xerox 1108 InterLisp-D, Smalltalk and Loops are powerful tools for developing the expert systems capabilities needed in CWS. Many vendors will be developing software for UNIX, Apple Lisa (Pascal), and Xerox environments. Portability of software should not be a major problem. Licensing arrangements will need to be negotiated early in the project.

Five-Year Forecast

The above 2 sections clearly indicate the technology for CWS will be available. Also, market factors indicate that the costs of hardware and software will be affordable. Briefly, if we examine Xerox, Apple and Masscomp's competitive positions we note that each company has many incentives to advance development of executive workstations.

First, Xerox is now heavily dependent on paper copier technology for sales and profits, but office automation and laser printers are making that technology obsolete. The Xerox AIS and laser printers are technologically state-of-the-art and it is likely management will recognize the threats and opportunities and move cash and other resources from copiers to marketing and developing executive workstations.

Second, Apple is promoting Lisa as a machine that is unique and far superior to other personal computers. They need to push and develop the technology to ensure corporate survial.

Finally, Masscomp has all of the advantages of a new company in a high-tech area: fast decision making, development, and enhancements. And all of the problems of establishing a reputation and competing with Apple and Xerox for the large potential market of executive workstations.

I expect a very competitive business environment for executive workstations with bold, efficient, imaginative management teams winning large initial market shares. IBM will likely enter the market within 2 years and the initial innovators will need to be well-established prior to that change. WWMCCS will benefit from this business environment because many development costs will be spread-out over business users.

Suggested Project Plan

Designing and developing a Commander's Workstation will not be an easy task . Management of a work group of 20 - 50 people, at various locations around the U.S. will be crucial to success. Finding "bright" graduate students will be necessary to hold down costs. A systematic plan and design methodology are indispensible. The experience of and advice researchers in the Decision Support Systems (cf., Sprague and Carlson, 1982; Bonczek, Hoisapple and Whinston, 1981) and Expert Systems (Hayes-Roth, 1983); also work by Charniak, McDermott, Pazzani, Schank areas will be invaluable in managing the CWS project and designing and developing new tools. Table present a plan and cost estimates.

Phases I and II in the plan (Table 1) could overlap in time to some extent. The activities under each phase are however interdependent and coordination is necessary. I do not believe that CWS can be developed successfully without the descriptive field studies in Phase I and extensive involvement of military personnel in all phases.

The initial commitment to the project must be from the very highest levels in DoD. These officer who make the commitment and their successors need to be directly involved in monitoring the project and they need to try early prototypes of CWS.

TABLE 1

ANALYSIS OF DEVELOPMENT ACTIVITIES, SCHEDULE AND COSTS

ACTIVITIES	LABOR	START	L, T, H COSTS*
PHASE I: Preliminary Study			$ 457,500
1. Prepare Detailed Project Plan	60 pd	1/1/84	30,000
2. Establish project advisory and evaluation group - 6 years duration	360 pd	1/1/84	200,000
3. Field studies describing commander activities	150 pd	3/1/84	200,000
4. Prepare additional scenarios	40 pd	6/1/84	20,000
5. Assess Phase I, revise Phase II plans	15 pd	7/1/84	7,500
PHASE II: Evaluation and Design			$ 452,000
1. Purchase and evaluate current hardware and software that has some of needed capabilities	180 pd	8/1/84	370,000
2. Design new software capabilities, prepare screens, flowcharts	1 py	8/1/84	60,000
3. Evaluate designs	30 pd	11/1/84	15,000
4. Revise plans-Phase III	15 pd	12/1/84	7,500
PHASE III: Prototyping			$3,365,000
1. Develop 10 prototypes of hardware systems using 1984 technology	100 pd	1/1/85	1,700,000
2. Develop software prototypes for various command situations and tasks	20 py	1/1/85	1,200,000
3. User testing of prototypes	250 pd	6/1/85	200,000
4. Refine prototypes	3 py	9/1/85	250,000
5. Evaluation of Phase III, revise Phase IV plans	30 pd	12/1/85	15,000

TABLE 1 (CON'T)

ANALYSIS OF DEVELOPMENT ACTIVITIES, SCHEDULE AND COSTS

ACTIVITIES	LABOR	START	L, T, H COSTS*
PHASE IV: Field Testing			$7,440,000
1. Develop actual hardware prototypes for various command situations	600 pd	1/1/86	1,400,000
2. Final software development and integration with external systems, networks, etc.	35 py	4/1/86	3,000,000
3. Final debugging, documentation, evaluation, etc.	35 py	10/1/86	3,000,000
4. Evaluation of Phase IV	45 pd	4/1/87	40,000
PHASE V: System Implementation			$2.25 billion
1. Competitive bidding	160 pd	5/1/87	100,000
2. Purchase 60,000-100,000 systems and needed optional equipment		8/1/87	1.5 billion
3. Systems installation		10/1/87	500 million
4. User and technician training		1/1/87	250 million

pd = person days

py = person years

* L, T, H total cost estimates includes hardware, travel and direct labor, but it does not include overhead which is estimated at 50% of the L, T, H cost figure. In my cost estimates, I am assuming some graduate students and post-doctoral researchers will be working on projects. Their time is not reflected in labor estimates, but it is included in the cost estimates. The costs of military personnel for their participation in all phases of the project are not estimated. Cost estimates for this project are in constant 1983 dollars. Cost and times indicated are based on a management system which minimizes bureaucratic delays. It may be possible to begin the project prior to 1/1/84. Also, adding personnel and greater reliance on high priced specialists may reduce project times, but increase costs.

Conclusions

In this paper I have attempted to provide a concrete image of a Commander's Workstation (CWS) and indicate its potential to improve military effectiveness and efficiency, especially in the realm of command, communications and control. In a recent paper (Power, 1983), I explored the impact of information management on organizations. Many of the issues raised in that paper are relevant in evaluating CWS (the paper is in Appendix II).

The analysis of technical requirements and current products indicates that it is feasible and practical to develop and have operational Commander's Workstations in approximately 60,000 command offices by January 1, 1989. The cost of hardware, software training and R&D should be approximately $2 billion dollars (in current dollars). This amount initially seems very high, but given that when CWS is operational it can improve our military effectiveness and potentially reduce clerical and staff costs by 25% per year (cf., Friedricks and Shaff, 1983), the product can not be dismissed as a luxury. Also, the long-standing Soviet interest in computerized command, communication and control systems raises the specter that they will exploit Western technological developments in this area before U.S. defense planners and politicians recognize the military advantage that has been lost.

REFERENCES

Bidwell, Shelford, World War 3, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978

Bonczek, Robert H., Holsapple, Clyde W., and Whinston, Andrew B., Foundations of Decision Support Systems, Academic Press, Inc., New York, 1981.

Druzhinin, V.V. and Kontorov, D.S., Concept, Algorithm, Decision (A Soviet View), Superintendent of Documents, U.S. Government Printing Office, Washington, D.C., 1972.

Friedrichs, Guenter and Schaff, Adam, Micro-Electronics and Society, Mentor Books, 1983.

Hayes-Roth, F. Building Expert Systems. 1983.

Pazzani, Michael J., "Interactive Script Instantiation", Proceedings of The National Conference on Artificial Intelligence, The American Association for Artifical Intelligence, 1983.

Power, D. J., "The Inpact of Information Management on the Organization: Two Scenarios. MIS Quarterly, September 1983.

Sprague Jr., Ralph H. and Carlson, Eric D., Building Effective Decision Support Systems, Prentice-Hall Inc., Englewood Cliffs, N.J., 1982.

Thierauf, Robert J., Systems Analysis and Design of Real-Time Management Information Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975.

Thierauf, Robert J., Decision Support Systems for Planning and Control, Prentice-Hall Inc., Englewood Cliffs, N.J., 1983

Williams, F., The Communications Revolution, (rev Ed.) Mentor Books, New York, 1983.

Druzhinin, V. V., Kontorov, D. S.
Concept, Algorithm, Decision (A Soviet View)
Superintendent of Documents
U. S. Government Printing Office
Washington, D. C., 20402
1972

Chapter 14. Automation Complex

Thoroughly develop fundamental and applied scientific research and introduce their results more rapidly in the national economy.

From Resolutions of the 24th Congress of the CPSU

1. Outline of Technology

We will try to imagine how the technology for utilizing an automated complex should look. The assumptions will contain a certain amount of imagination and the description by necessity will be fragmentary and extremely simplified.*

The commander's order of operations will not necessarily coincide with the order described below. Certain functions may seem to be unnecessary, certain others will be altered, and new functions will appear. We are interested in the use of automated means, and only from this point of view will we examine the commander's work.

The commander arrives at a new assignment and, after becoming familiar with his deputies and closest associates, he embarks on a task of situation analysis. All the required factual data can be obtained from the data retrieval system (DRS). Data about the grouping of forces in the area of combat operations are displayed on the general situation screen of the command post: first in small scale: the enemy, our forces, neighbors, communications; and then in larger scale: the enemy disposition of troops, lines; fighting strength; and finally in large scale: enemy strong points, his weapons, state of communications. Documents are requested, which contain quantitative data on the makeup of enemy units, characteristics of his military equipment, military experience, and morale, as well as lists and combat characteristics of the command staff. If there are no data on certain questions, these questions are directed to the intelligence branch for the collection of additional information.

The commander then undertakes *initial familiarization* with his own troops. Using the DRS and interaction with his closest aides, he becomes familiar with the disposition, makeup of troops, weapons, equipment, data on combat capability, combat readiness, political-moral state of the personnel, and the combat, political and psychological characteristics of commanders directly subordinate to him. Some of the questions for which there are no data are directed to subordinate troops and staff sections for clarification. Formulation and description of the present and the future missions of higher headquarters are displayed (on the *indicators*). Then the commander may become familiar with previous events. If necessary, the course of combat actions in a given region can be reconstructed on the screen (in convenient time scale). Proposed, but unimplemented decisions are noted. This phase of operations ends in familiarization with current combat orders and instructions, logistics, and contact with subordinate commanders of various ranks and neighboring units (automatic communication and data display are used here).

The next step is to analyze the theater of military operations (terrain). Photographs of various battle areas and the surrounding sectors are displayed by stereoscreen. By changing the scale and foreshortening, the commander can acquire the most informative representation of the data, which are simultaneously supplemented with digital data and text. The commander and his aides analyze the terrain, weather situation, geophysical conditions; using highly informative means of interaction he asks questions and receives answers. The subordinates, in turn, direct the commander's attention to details which they think are important. The mission assigned to the troops is shown on large-scale stereoprojections of the terrain.

This phase is concluded by posing a number of questions, the answers to which require a computer complex. These questions basically concern possible variations of combat actions, logistics, preparatory measures and their organization. The purpose is to concentrate on those questions which arose in the commander's mind as a result of his initial impression. Some of the points are specific in character: prepare mathematical data, evaluate the effectiveness of certain measures and actions. Other points are more general: propose solutions for problems and subproblems, forecast the situation, evaluate the enemy's intentions. Even the most general points must be distinctly formulated with specific limitations, in a quantitative statement if possible: suggest solution variations of a rigorously formulated problem, submit a forecast and intentions relative to certain factors. There should not be questions such as "What does the enemy intend to do?" Instead, ask "What is the probability that the enemy will attack at sector A within the next 2 days?" The answer may be approximately as follows: "For the next 2 days the enemy may concentrate such and such forces in sector A; the effectiveness of attack is such and such, the probability of attack is such and such."

Having formulated the questions and designated the time that he wants the answers, the commander may visit his troops for on the spot familiari zation with the situation. In this stage of the operation the automated systems of subordinate units are used.

In the next step, the commander returns to his command post with new ideas and impressions, and begins to make decisions. But first he listens to and discusses the answers to questions previously asked (including new questions that arose during his visit to the troops). He makes his initial information decision relative to the situation, state of enemy troops and his own troops. On the basis of the information decision (which is passed on to the subordinates) and mission, assigned by the senior commander, he formulates his *initial* ideas of an operational decision. The commander briefs headquarters on the overall objective of the forthcoming battle and conducts discussion. Automated systems make it possible to recruit a supervisory staff from the field. The suggestions are fed into the computer for evaluation and utilization by headquarters and services for the preparation of their proposals.

The next step may be characterized as the discussion stage between the commander, staff and services. Background reports are prepared; the computer evaluates the proposals made during the "brainstorm," analyzes them, separates the constructive ideas and uses them for preparing alternatives of a decision. Key personnel formulate their own ideas and proposal, and feed them into the computer for subsequent combined analysis.

The most important step is decision making. It may begin with an examination of alternative decisions of the computer, which are displayed automatically for review. Each alternative is accompanied by a list of positive and negative features, and effectiveness evaluation. Discussion of the alternatives includes indication of weak spots, alteration of limitations and input of additional data into the program. The discussion is conducted with the aid of highly informative means of interaction, and the DRS records new proposals. As a result, some of the alternatives are discarded, some are improved, and new alternatives are developed. The discussion continues until only one alternative is left, which is approved, or else the commander selects one of the alternatives, alters and improves it (using the automated complex) until he considers it to be the best one.

The decision is sent on to headquarters for detailing. One aspect of detailing consists in mathematical modeling of the forthcoming battle as a whole, of its elements and individual logistical aspects. Modeling makes it possible to consider the influence of random and secondary factors that escape the field of view during general examination, and also to evaluate the effectiveness of the designated measures.

We have examined one rather arbitrary version of the decision making process. All other areas of technology should implement the principle of allowing commanders and operators to spend the maximum time and effort for creative work and direct command of troops. Otherwise, it is impossible to implement the directive of the Minister of Defense, Marshal of the Soviet Union A. A. Grechko:

"The commanders and headquarters at all levels will creatively solve problems of combat readiness, and concentrate their attention on the most important and long-term trends on which depend our superiority over a probable enemy. Their thoughts and efforts should be focused on the search for new capabilities and alternatives for continuously increasing the fighting power of the Armed Forces."[1]

2. The Consultant [Konsul'tant]

As seen from the material presented above, the information functions of the commander and staff play a great role in decision making. The collection, selection, systematization, interpretation and presentation of data require a great deal of time and effort. These are consultative functions and they can be automated. An electronic consultant, depending on the organizational level, may have different dimensions. Judging by known foreign models, a rather large DRS can be housed in one rack with a desk of ordinary dimensions and may encompass transcription and operational reports. A useful DRS with low information capacity is easily made portable (in a field pack). A small DRS, containing an electronic scratch-pad memory, retrieval computer with display and push-button programmed control, is quite adequate for current operational work and may become a reliable "constant companion of the commander." The small DRS should be connected periodically to a large one (directly or through a communication channel). It is essential that the commander personally (and not through delegated persons) use his own DRS, change programs and monitor the informational completeness of his consultant, treating it as a personal weapon, as a means of expanding his own memory and sensory organs. Only in this case can it be effective. Other key personnel may have their own small DRS of the same design, but with professionally oriented information.

DRS, like people, should interact with themselves and with people in order to understand each other and continually renew their information resources. DRS are easily made to "forget" (much easier than man) unnecessary data. They readily receive new data, but continuous monitoring of this process is required.

A considerable advantage of the electronic consultant is the fact that it can be entrusted without danger with random thoughts, instantaneous ideas and considerations that appear promising; it does not distort or forget them, does not confuse the address and stores them until they can be developed, used or discarded. The consultative function of the

[1] A. A. Grechko, *Na strazhe mira i stroitel'stva kommunizma* (On Guarding Peace and the Building of Communism), Moscow, Voyenizdat, 1971, p. 58

automated complex should embrace all aspects of activity. When we speak of an automated complex we do not simply mean the DRS alone. We are talking about the entire set of automated systems that support military organizations. If only the commander has a DRS there is little to be gained: an isolated island of automation is nothing more than an exotic entourage in the complex technical equipment of an army.

The strength of automation lies in the complex, the systems approach, and in interaction and mutual information. The memory of DRS is limited, regardless of the level of microminiaturization, both technically and in terms of content, since this memory is made up by the people who use the DRS. But life is always deeper and more complex than can be foreseen, especially by one man. In judging collective activity we established that the development of complex problems requires a collegial structure. This conclusion is also valid in relation to automated systems: a system of compatible and constantly interacting DRS is very effective.

The consultative functions of the DRS should not be limited to reference functions. The computer of the DRS should be used for doing calculations related to the display of information and evaluation of effectiveness. The volume of operations which can be carried out in this regard by the retrieval computer of a small DRS is low, but the requirements here are correspondingly limited. The position of the electronic consultant in the decision making system is illustrated in Figure 72. Here the operators are released from reference work and calculations; this increases their creative capabilities.

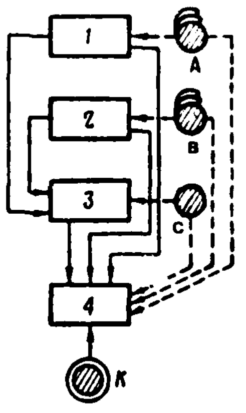
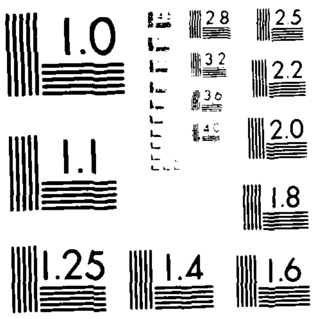


Figure 72. Diagram of an electronic consultant: 1—reception, processing, and presentation of data on the enemy and operational conditions; A—reconnaissance operators; 2—reception, processing, and presentation of data on friendly forces; B—operators; 3—display of combined information on the situation; C—situation analysis and preparation of alternative decisions operator; 4—effectiveness analyzers and comparison of alternatives; K—commander.

AD-A142 570 WIS IMPLEMENTATION STUDY REPORT VOLUME 3 BACKGROUND INFORMATION(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T H PROBERT 01 OCT 83 IDA-D-51-VOL-3

5/6

UNCLASSIFIED IDA/HQ-84-28344 MDA903-79-C-0018 F/G 17/2 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

This plan is based on the utilization of automatic data collection and display systems by operators. Interaction between the operators is based on evaluation and improvement of the decision alternatives developed by them. The main operational function of the computer is to evaluate the effectiveness of the alternatives. It is assumed that reconnaissance data and data about our troops will be processed by different operators. The combined information about the situation is formed on the basis of data selected and appropriately processed by the operators and the computer. This information is used by the operator who prepares decision alternatives. The commander may not only correct the decision alternatives, but also feed into the computer other alternatives which will be evaluated. Thus the DRS not only gives information, but also develops new *information (estimates)*.

It is often necessary in military practice to evaluate many alternatives, each of which may be described in sufficient detail, but in view of the very detail of description it is necessary to resort to unwieldy calculations in order to arrive at an evaluation. Such a "rash of alternatives," when "alternative after alternative was proposed . . . and after heated discussion was thrown out,"[1] is described by Marshal of the Soviet Union A. A. Grechko. During the Great Patriotic War, one could not determine whether or not various alternatives were "unrealistic, both from a military theoretical, as well as a practical, point of view."[2] There were not enough calculations that could be done only by the simplest means and they were not good enough for making definitive conclusions. It took a great deal of time, effort and valuable talent on the part of high ranking military chiefs to arrive at such conclusions. The electronic *consultant* not only eliminates these worries, but just as important, it provides the foundation for thorough examination of plans of action, careful situation analysis, determination of obstacles and ways of overcoming them.

3. *The Assistant [Pomoshchnik]*

The electronic consultant does not perform decision preparation functions, let alone decision making functions. In order to help the commander and his staff in the performance of these functions, it is necessary to develop a computer section and means of interaction between the automated complex and the corresponding control links. Then the electronic assistant will be capable of independently working out proposals and justifying them. The decision to adopt or not to adopt these proposals is the responsibility of the commander or other key personnel. Proposals may pertain primarily to information decisions. Control of the parameters of the information decision preparation program (*input* of

[1] A. A. Grechko, *Bitva za Kavkaz* (Battle for the Caucasus), Moscow: Voyenizdat, 1967, p. 242.
[2] *Ibid.*

weight coefficients for various sources of information, limitations, etc.) is the responsibility of the operator, but all data processing and evaluation of the reliability of decision alternatives are entrusted to the electronic assistant. Proposals may also pertain to organizational and operational decisions, but we may speak here only of certain fragments, and not of complete decision alternatives. The alternatives of operational and organizational decisions (as well as the making of information decisions with consideration of the computer alternatives and their justifications) are developed by the operators. The effectiveness evaluation and optimization are performed by the automated complex.

The programs and data of the "assistant," to a greater extent than of the "consultant," are individualized and specialized in accordance with the personal features of key personnel, character of the groups, and general arrangements made within a given group. The "assistant" requires more continuous combat evaluation, supplementing of programs, revision of old data, and continuous direct interaction. Cooperation between people and machines, just as between people at headquarters, is essential.

The development of automation is aimed at the reassignment of information, computation and evaluation problems to computers. If an electronic assistant is available, the commander and the operators may direct almost all of their efforts into the creative channel since they have all the necessary data for this purpose and are not distracted by secondary problems.

The position of the electronic assistant in the decision preparation scheme is illustrated in Figure 73. The electronic assistant is assigned additional functions of working out information decisions and optimization. The alternatives of operational and organizational decisions are prepared by the operators. They control the actions of the electronic systems and can actively intervene in their work; the extent and the result of this intervention are recorded and are made known to the commander in order that he can know exactly what aspects of the situation were contributed by the operators. Through electronic systems, the commander may influence the work of the operators, suggest ideas to them and cooperate with them in any project. Reliability evaluation, along with effectiveness evaluation, is not the concluding, but an intermediate result of the work, the guiding factor and stimulus for improvement of the decision. The functional structure of combat evaluation is continuous here, i.e., the operators theoretically can work without the aid of the computer (if they have the know-how). But the advisability of an action, then reliability evaluation or effectiveness evaluation will indicate information or operator B proposes an infeasible or irrational alternative actions, then reliability evaluation or effectiveness evaluation will indicate this. Control is not absolute; not all errors are detected since the program does not guarantee coverage of all decisions which man is capable of thinking up. But continuous improvement of the programs should minimize such cases. In this regard creative thought should be developed in consideration not only of the features and characteristics of subordinate key personnel and groups (to which we are all accustomed), but also in consideration of the features of the electronic assistant. The utilization of the electronic assistant does not merely simplify and facilitate work, but also enriches it with new qualities and possibilities. Alternative decisions are developed by people (except for information decisions, where this is not mandatory), but their evaluation (and consequently their quantitative justification) and the improvement that can be achieved within the framework of the logic employed, are entrusted to the computer. The system cannot operate independently (without operators); its functional structure is fragmented.

It should be recalled that an automated complex embraces not one command post, but rather a system of interconnected command posts; in this regard, the structure illustrated in Figure 73 should be related to other analogous structures. An informational connection is required here, and not an operational one: compared with the "consultant," the "assistant" should have not only larger computers, but also permanent lines of communication with the corresponding transmission capacity and high reliability. This does not mean that it is useless or impossible

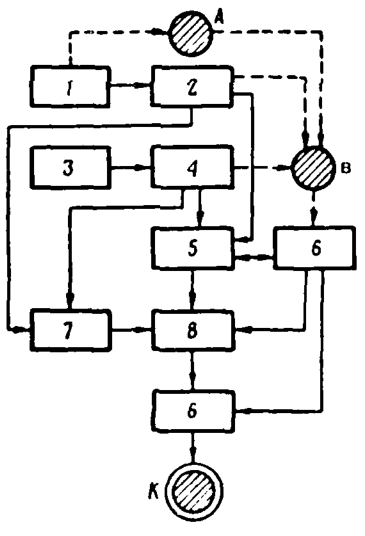


Figure 73 Diagram of an electronic assistant 1--data on enemy, 2--development of informational decision on enemy, 3--data on friendly forces, 4--development of an informational decision about friendly forces, 5--optimization, 6--presentation of data; 7--reliability evaluation, 8--effectiveness evaluation, A, B—operators, K--commander.

to automate only one or a few command posts (and not all at once). However, the full effect can be achieved only through a computer complex because we are concerned not only and not so much with the convenience of operation as with the quantitative evaluation of an idea and the essence of a decision; this can yield the greatest benefit if the evaluation is done in all interconnected links because it is impossible to perform the entire volume of work in one (even higher) link. The lower-level links of the control system can be equipped later with systems that convert the "consultants" into "assistants"; the reequipping process may take a long time. But after it is completed the "assistant" of the higher-level link may be used in the entire system of interconnected command posts. This is desirable since it enriches computer programs and encourages interaction. The automated complex is constructed by the hierarchical principle, and all the operational information circulates in this structure. This ensures the preservation and total utilization of the data. The structure should be highly reliable and viable.

Combining the "consultant" and "assistant" in one complex gives the operations staff time for clear thinking after having requested information and having loaded the computers with work.

The electronic assistant gives complete analyses and accurate evaluations.

Analysis must be especially detailed and thorough when new forms of weapons are used or when weapons are deployed under atypical conditions. An evaluation of only already-available alternatives is not enough here: it is necessary to find weak spots, and to aid in the determination of the directions of future creative search. Analyzing the Novorossiysk offensive operation during the Great Patriotic War, Marshal of the Soviet Union A. A. Grechko emphasizes the joint ground and naval operations. This operation may be regarded as a complex situation. Here is how A. A. Grechko describes this situation:

"Thanks to the accurate artillery fire, it was possible to destroy the enemy's engineering fortifications. Powerful artillery bombardment made it possible to land forces in the port of Novorossiysk quickly and without great losses.

"The intelligent combination of the elements of surprise (with respect to time, location and extent of the front of the landing forces) and the application at that time of a new method of massive deployment of torpedoes against coastal installations and fortifications stunned the enemy, scattered his forces and prevented him from quickly organizing a strong counteraction at all points.

"The Novorossiysk offensive operation had several important features. Thus, the dispersion of the forces of the 18th army of Tsemesskaya Bay, limited access roads and directions, and the small areas of the initial regions dictated the choice of the directions of the main and supporting strikes. These same circumstances influenced the composition of forces and equipment required for carrying out the *operation*."[1] The Novorossiysk operation verified the fact that all branches of services may be used in moderately rugged mountains and large cities.

No less characteristic in this regard was the activity of the commanders of the individual groups that carried out the combined operational mission, but which were located in tactical isolation. The leadership of such groups required the ultimate utilization of creative abilities. Analyzing a similar complex situation, Marshal of the Soviet Union A. A. Grechko wrote: ". . . The army commander (he is speaking of the actions of the 56th Army in *1943—V. D. and D. K.*), on 17 September ordered the troops, pursuing the retreating enemy, to organize attack groups in the main directions. Their mission included: penetrate the enemy's defense at his intermediate positions and by wedging into the rear, cut the enemy's escape route and destroy him unit by unit. The army commander, taking advantage of the fact that the enemy did not have a continuous front, ordered mobile detachments and machine gun groups to courageously penetrate to the enemy's rear with the mission of creating panic in the enemy's defense, and paving the way for the advance of strike groups in the main directions."[2]

In situations of this type the electronic assistant may be an indispensable means of creative interaction among commanders and of operational cooperation among them under conditions of an uncertain situation, dispersal and surprise.

4. The Comrade-In-Arms [Soratnik]

The electronic assistant is not capable of independently proposing (let alone making) operational and organizational decisions. Figure 74 shows an automated complex which performs the entire sequence of decision preparation decision making functions under the continuous control and with the participation of the operators who can use the computer results in any stage, feed in new ideas or corrections, but who are not required to participate in the development of the *computer decision*.

The diagram shows three channels: two resolving and one teaching. One of the resolving channels is an automatic computer channel, and the other is a "heuristic" operator channel. The computer channel analyzes the input information and information decisions, prepares alternatives of operational (organizational) decisions, selects criteria, evaluates effectiveness, and optimizes a decision.

The operator channel performs the very same functions. Crossed inter-

[1] A. A. Grechko, *Bitva za Kavkaz* (Battle of the Caucasus), Moscow, Voyenizdat, 1967, p. 379.
[2] *Ibid.*, p. 383.

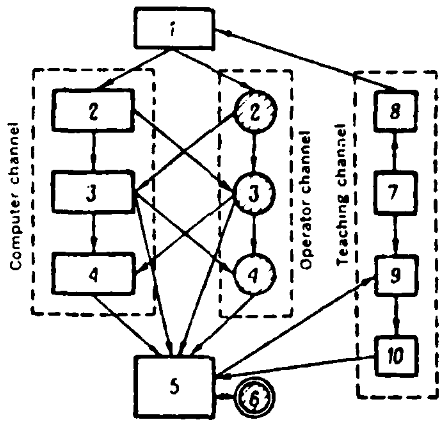


Figure 74. Diagram of an electronic comrade-in-arms: 1—input data; 2—preparation of informational decision; 3—preparation of alternatives of operational (organizational) decisions; 4—evaluation of effectiveness; 5—display of decision; 6—commander; 7—preparation of problem; 8—preparation of input data; 9—analysis of results; 10—recommendations on teaching.

action is provided between the resolving channels: the output of each unit is connected to the next unit, both of its own and of the next channel. The operators can use all means of automation for consultation and assistance, but the decisions are worked out manually. The final decision is made by the commander in consideration of, or on the basis of, the results produced by both channels: "willful actions" in the computer channel are replaced by "threshold action," comparison of the output values with the thresholds or of several values. The thresholds are established ahead of time, but their values may change, depending on the results of the operation of the operator channel. Consequently, the computer channel carries out a willful action, formulated ahead of time or during the operating process. The teaching channel is designed for building a thesaurus in the channels and for training them in problems of increasing complexity. The teaching problems should consider new achievements in military science, the requirements of practice and the future. Teaching includes the formulation of such problems, analysis of solutions, disclosure of deficiencies, development of instructions to the combat team at the command post, and introduction of changes in program. The operators of the second channel cannot be given this function: performing the analogous function, they inevitably would insist on their ideology and methodology, alter the computer channel to their liking and eventually transform it into their own pale copy. The special group, assigned to teach the system, will teach and improve itself, discover new, often unexpected results, find the reasons for their appearance and think up new problems.

The three-channel structure ensures the independence of formalized (traditional), and intuitive (creative) methods of decision preparation and teaching. The basic concept is to ensure, in any case, a timely workable decision, and if an original, creative decision is worked out, to consider it also. The combined analysis of the decisions worked out by the two channels can stimulate a more effective decision which the commander proposes. The teaching group is very important. Its role consists not only in the continuous correcting of programs and training operators, but also in the implementation of a certain operational ideology, organization of improvement, cooperation and mutual stimulation of the algorithmic and heuristic channels.

The main advantages of the system are mutual stimulation of the channels and mutual control. The operator can propose the most improbable. decision without risk of consequence: everything is subjected to at least a double check. Competition between the channels and the presence of different alternatives suggest new ideas to the operators. The interaction of the channels reduces the decision preparation time. The automated complex, embracing the entire system of command posts, does not have to contain electronic comrades-in-arms in all links. Perhaps at a certain stage it will be necessary to have "consultants" in some (obviously low level) links of control, "assistants" in higher-level links, and "comrades-in-arms" in the highest and most important links. The use of "comrades-in-arms" in lower-level links at the present stage of development of technology is fraught with enormous problems in the design, adjustment and improvement of a multiconnected system that includes people; a system which must operate in a stressful situation with an acute shortage of time. These, however, are temporary problems.

With high information communication channels, the electronic comrade-in-arms may service (at least through the computer channel) lower-level organizations. Therefore, the automated complex as a whole expands its its comrade-in-arms functions to all organizations, in spite of the fact that the technical equipment of the lower-level control links may remain at a lower level for a long period of time. It is difficult to predict the future competence of the electronic comrade-in-arms and how great an influence it will have. It is clear, however, that a workable decision is always ensured, and that the creative energies of the commander and his staff will be liberated to the maximum extent from technological functions. Teaching and self-teaching of the "comrade-in-arms," expansion of its thesaurus and programs will be accompanied by a general improvement of means of automation and development of group intellect

The Impact of Information Management on the Organization: Two Scenarios

By: Daniel J. Power

Abstract

A concept called information management has been discussed for many years by computer and management scientists. Implementing this concept may revolutionize organizations and have a profound effect on organizational decision making. Since the technology needed to implement sophisticated information systems is now available, managers need to address the potential impact of this innovation on their organizations. This article presents two scenarios that may help managers to anticipate the effect of information management on organizational decision making.

Keywords: information management, organization design, decision making

ACM Categories: H.4.0, J.1, K.6.1, K.7.1

The author wishes to acknowledge the comments of Barbara Armstrong, Anne Fair, Alan Hevner, Judy Sorum, and an anonymous reviewer. Jerald Hage and Paul Collins made helpful suggestions during a workshop presentation sponsored by the Center for Innovation, University of Maryland.

Most organizations do not have sophisticated information systems despite optimistic predictions about integrated management information systems (MIS), relational databases, and the growth of data administration [6, 18, 20]. Many predictions about the growth of information management may have been overly optimistic, but not necessarily wrong. Improved management of organizational information may yet revolutionize organizations. Through continuing advances in hardware and software development, the means are now available to implement extremely sophisticated information systems. Therefore, managers must contemplate the consequences of expanding information management activities [22].

This article examines changes in organizational decision making that may result from innovations in information management. Information management is used here as a broad term that includes data management and data dissemination activities in all parts of the organization. While the terms information and data management are sometimes used interchangeably, most users of information management assume that data are shared by different organizational units, that an overall view of the organization's data needs exists, that data are controlled and synchronized, and that redundancy is minimized [5, 10, 14, 15, 19, 21].

Organizational decision making includes important individual and group activities of problem identification, information search, evaluation/choice, and the implementation of actions [3, 17]. Organizational decision making activities occur in a structure of managerial roles and responsibilities [16]. Following Anthony this structure can be conceptualized as having three primary levels — strategic, managerial, and operational [2].

Two alternative scenarios are presented to provoke both managers and researchers to more closely examine the consequences to organizational decision making of attempts to better manage information. Focusing on organizational decision making is important because, as the scenarios demonstrate, decision activities and structures may be altered radically by changes in information management. The scenarios present contrasting views of how sophisticated information systems can be implemented. In Scenario 1, decision making responsibilities and organizational design are drastically altered as control is centralized and the organization begins to rely heavily on an integrated information system. In Scenario 2, the decision making structure and structure remain relatively traditional and a very heterogeneous information system including traditional communication media like newspaper clippings and internal memos is used.

The article may have an implicit bias toward Scenario 1 because it is intended as a projected state-of-the-art description. The managers in that situation rely heavily on computer oriented techniques to manage information. One must, however, be cautious in concluding that the description in Scenario 1 is best in all situations. The Scenario 1 outcomes are probably desirable only in certain types of organizations. For example, a more centralized, integrated information system may be feasible and effective when the divisions of an organization have similar products or services.

Using Scenarios to Understand the Impact of Information Management

State Scenarios "posit what the world or relevant context will be like a number of years from the present, without describing at the same time how the world 'gets to be that way.' Process scenarios, in contrast, specify the sequence or chain of events that lead up to a particular future state" [9].

Both process issues and future states can be used to help analysts and managers deal with details and dynamics that are otherwise easily avoided. Such scenarios can illuminate the interaction of multiple variables; they can present issues forcefully by simplifying the model of the organization or system; and they can aid in the consideration of alternative outcomes [12].

Scenarios are useful because information technology and managerial attitudes are changing so rapidly that it is difficult to predict the direction and magnitude of changes in organizational decision processes and structures based on trends. One reason that such prediction is difficult is the ambiguity about how database technology, a key factor in *information management*, will develop. For instance, Robinson suggests that database *use in organizations* is so *dynamic* that both the first *and* second derivatives of change are positive [21].

Other factors that complicate such predictions are the doubts, fears, and antagonisms of some managers toward information technology and most likely toward information management as well. Leavitt and Whisler long ago raised the possibility that the introduction of information technology would drastically reduce the number of middle managers [13]. Blumenthal [4], Dearden [7], and Ackoff [1] attacked what they called the exaggerated claims made by promoters of management information systems. Both of these factors, change in technology and managers' responses, remain difficult to evaluate and account for in a forecast. The first factor suggests an optimistic forecast, while the second suggests a pessimistic one.

Scenarios can also help managers evaluate the benefits and costs of proposed information innovations. They may as well provide longer lead times for planning and for shifting physical and human resources to alternative uses.

The two information management scenarios

The two scenarios take place in a hypothetical multi-division conglomerate. Although the scenarios deal with a profit-making organization, many parallels can be drawn with other bureaucratic organizations [11]. In both scenarios managers have the goal of implementing information management. Controllable factors such as resistance to change and the knowledge of users are assumed to be facilitative rather than inhibiting. Each scenario moves from 1983 to 1989-1990 at which point a "snap-shot" of organizational decision making activities is presented. In both scenarios, the different implementations of information management can be seen to have had a significant impact on organizational decision processes and structures [8].

Four technological and administrative variables related to information management are examined in the two scenarios power of the data administrator (high/low), sophistication and centralization of the database (high/low), sophistication of data entry and output (high/low), and systems control (high/low) Table 1 summarizes the assumptions about each of these variables for the two scenarios In many ways the two scenarios represent extreme, but plausible states for organizations in 1990 Such a focus can often present issues more forcefully and distinctly

Scenario 1

In 1983 the XYZ Company, a conglomerate, began to reorganize and plan for the introduction of information management activities The first step was the appointment of a data administrator. The person selected for the job had control over all information resources of the company, e.g., access to historical files, creation and control of new information; and most people at XYZ knew the administrator had a major role in making all company decisions. This individual assembled a staff and planned the integration of data collection and storage

The second step was to develop information gathering and storage standards for the entire company In 1984 the third step was begun when duplication of data was evaluated and actions were taken to eliminate redundancy At approximately the same time the organization purchased a new database management system (DBMS) with powerful relational characteristics Also, the company greatly supplemented online storage capacity at the central computer center

Beginning in late 1984 the company added more mini and micro-computers and created a distributed processing network with a large centralized memory capability In the next step most data collection was put online Also, more automatic sensors and recording devices were used, especially for production and sales Finally, in the fifth step many decision tasks were routinized and programmed. Routine decision making programs were developed to interface

Table 1. Summary of Two Scenarios: Information Management Variables

	Variable Ranges	
Variables	Scenario 1	Scenario 2
Data administrator	A "strong" administrator at executive VP level, with separate budget.	A "weak" administrator, committee coordination, many managers have DP budgets.
Database Management	High level of control, conceptual integration, data independence, e.g., relational database.	Low level of control, redundancy, little integration of data, many databases.
Data Entry/Output	Company wide procedures, online and sensor entry, graphics, work-stations, projection TV.	A variety of procedures, some online and some paper forms requiring batch entry, emphasis on reports, much staff intervention required.
System Control	Central control and standards, networking, data sharing in system.	Separate, autonomous systems, no uniform procedures, no data sharing in computer system.

with the DBMS. Also, managers were trained to use the database. By 1989 the fifth step was completed.

As a result, in 1989, the three members of the Office of the President (OP) of the company meet at a weekly assessment session. The Office of the President includes the president, the vice president for information and financial analysis, and the vice president for products and marketing. The vice president for information and financial analysis controls all information resources of the corporation.

At this weekly meeting the president's usual procedure is to review all exceptions from the weekly status reports. For example, production exceptions are evaluated for divisions with production overruns or shortages greater than 10% of established targets. The procedure for evaluating weekly exceptions usually includes a quick search of the management database to see if any relevant reports were filed. If an explanatory report is found for a discrepancy, the executive summary is usually flashed on the display screen in the front of the conference room and the three officers quickly scan it for any important information. Usually some follow-up activities are specified for exceptions even if there is already relevant information in the management database. The standard procedure is to request the division officer to identify any information in the database relevant to the exception and to provide additional information. That request is transmitted using the information system telemail. The division officer normally responds very quickly with a short telemail memo noting document numbers or memory location identifiers. The memo is stored in a current messages file by division and topic. The division officer also is responsible for the accuracy and completeness of information stored in local databases and the management database. All division exception reports are handled in this way.

Following a review of exception reports, the three executives usually do a search of external databases and information in the management database to identify any recent changes that might be relevant to the company. In preparation for these meetings, detailed searches of external databases have already been conducted by the planning staff and that information is in the management database. The officers quickly run through the titles or summaries of the relevant items. During the screening each member has the opportunity to stop and display and request more detailed information. The information is preselected, but the officers also often search for, read, and discuss other materials about the company's external environment at weekly assessment meetings. They may spend as much as five hours per week reviewing information sources and discussing the changing environment of the company.

Following the status analysis and the analysis of environmental information, OP members make an assessment of long-run trends that may create threats or opportunities given the firm's strengths and weaknesses. At monthly and quarterly meetings assessment files are re-evaluated to determine if an additional information search by the planning staff should be conducted. Also, at the monthly and quarterly meetings the officers use decision aiding programs to select data gathering procedures, to develop decision trees and to structure their decision making.

Many other decision makers in XYZ also use company databases. For example, division officers store information and search the integrated databases to find out about new programs or activities taking place in other divisions if material is authorized and cleared. The division managers often use the information in databases as part of simulation and forecasting programs for production, marketing, and personnel. Also members of their operating unit staff often access the databases.

Plant and service managers use a database for the daily planning of schedules, evaluating absenteeism and tardiness problems, and checking on inventories and orders. For plant and service managers, most of the clerical and information recording activities are handled automatically and recorded in a local database that is linked to the management database. For example, when time clocks directly record information into a database, that information is immediately available to managers on an exception basis to help enforce disciplinary policies or to handle other discrepancies that may be noted by management control computer programs. Many decisions of the plant and service managers are very routinized. Programs provide exception reports, help in scheduling, and in handling inventory management. Plant and service managers have significant time to actually observe activities, talk with employees, and check on maintenance crews.

Most of the accounting and finance activities are also routinized. Cost accountants and financial planners report to the vice president for information and financial analysis as well as to division managers. Many sales functions are automated, but salespeople in divisions continue to service customers and they occasionally enter orders.

Scenario 2

In 1983 the XYZ Company began to plan for the introduction of information management in the organization. The first step was creation of an information management committee. This committee included technical people and division managers. All information projects were reviewed by the committee. The division managers continued to decide what resources would be used for data processing projects and had to specifically propose each project. In 1983, the committee published a project booklet and developed guidelines for uniform data collection. Then, this material was disseminated to all division managers and systems designers.

In late 1984, the company upgraded hardware and software in some divisions. Two advanced DBMS with a hierarchical structure and a simple query language were purchased. Then, divisions were permitted to create and revise databases and some divisions chose to do so. The corporate planning staff also developed a planning database using micro computers in 1986. Also, in 1986 the planning staff went completely online for data entry and each staff member used a terminal regularly. In 1986, some division managers purchased sophisticated distributed processing equipment and automated some data collection.

The next step was an attempt to provide uniform management reports. Each division supplied information regularly to the central planning staff. Finally, the central planning staff developed exception report programs in 1988.

As a result, in 1989, the president (CEO) of XYZ Company scans weekly status reports from each division. The CEO tries to identify any exceptions that occurred during the week. Members of the planning staff underline any potential exceptions. The staff prepares a list of status exceptions for the president who then tries to assess what each exception means, and requests more information from division officers if the problem seems serious. The CEO also usually makes notes on the list of exceptions, sending the annotated list back to the planning staff, who keep track of possible exceptions in the planning database. The planning staff also searches for trends and enters the president's comments into the planning database prior to the next weekly assessment.

The president works closely with corporate specialists in finance, marketing, personnel, operations, and information systems, and has many contacts with group and division managers. The planning staff is very active, but most of their work is involved in tabulating reports from the divisions and keeping the planning database current.

At the weekly assessment, the president likes to determine what is "going on in the world," taking account of information in key newspapers and information received from a clipping service. At the weekly assessment the planning staff usually presents five or six folders of clippings. The president scans these articles, sometimes asking for more details or follow-up from the planning staff and asking the planning staff to keep track of any potential threats to the company. On a regular basis, the file of possible threats and opportunities to the company is reviewed by the CEO and advisors on the planning staff. This occurs at least once a year and usually there is an informal conversation about these issues.

Information in division databases is reflected in reports and summaries provided to the president but the CEO really does not have direct access to that information. At the division level, each division officer has access to a database although they do not all have direct access through a query system. Each division officer has a programming and support staff that can respond to ad hoc requests for reports in a day or less. In some divisions, managers use division databases quite frequently to check on the status of operations. Some management control programs have been written to identify exceptions using the databases but the dissemination of these programs to divisions is limited. Part of the reason for this is that divisions have different data structures, so that it is difficult for a manager in one division to [illegible] a program that can easily be used by another division

XYZ has a division manager "sharing system" and every month division managers contribute information about activities in their division. Some managers store that information in a database. Then, if in the future they need more information, they get on the phone and talk to the manager in the other division and request more details.

At the operating level, the plant and service managers are somewhat involved in scheduling and planning, but most of those activities are handled by computer programs. Problems with the quality of the data (from nonstandard procedures) usually require that managers recheck all the major decisions made by programs and sometimes, verify information used by the programs. Some of the operating units have better information collection procedures than others since some units continue to use people to enter data and others have data collected automatically by recording and sensing devices. Employees in most units follow multiple data handling steps to get the information into computer databases. In each division the databases at the plant and service level are usually quite standardized, and for most of the plants and service centers the database is sufficient. Some of the databases designed by XYZ's various divisions are more efficient and flexible than others. The most effective can be tailored to meet the needs of different units within a given division and yet allow for the sharing of information among different units.

The decision structure at all three management levels is still very traditional as it was in 1983. Also, most organizational information is distributed by verbal and written media, rather than computer systems.

Analysis of the scenarios

In both scenarios, the implementation of an information management system led to many changes in XYZ (see Table 2). But the most dramatic changes developed in Scenario 1. Decision making responsibilities and organizational design were drastically altered. For example, a vice president for information and financial analysis joined the company's president and the vice president for production and marketing to create an Office of the President. Also, in Scenario 1, strategic decision making was more systematic

Table 2. Summary of Possible Consequences

Scenario 1	Scenario 2
More systematic and more collaborative decision making at strategic level	More information at the strategic level may slow decision making
Reorganization of decision making responsibilities at all levels	Division officers make more operating decisions
Greater control at strategic level over operating level	Some division officers have greater control over operating level.
More programmed decision making delegated to computer	Some programmed decision making at operating level using databases.
Information sharing is facilitated.	Little sharing of information between divisions.
Increased span of control is possible because of the reduction of number of decision makers and support staff	More staff people needed to process increased information.

and collaborative, and programmed decision making was much more prevalent at all levels of the organization than in Scenario 2. XYZ company required fewer decision makers and fewer levels of decision making in Scenario 1 than in Scenario 2. In Scenario 2, a large addition was made to the decision support staff at strategic, managerial, and operating levels.

In both scenarios, the organizational control system was altered, but in Scenario 1 the Office of the President exercised more direct control over the division and operating level than in Scenario 2. The quality, timeliness, and amount of information increased in both scenarios, but in Scenario 1, XYZ company provided managers with numerous automated aids for information retrieval, display, storage, and decision making.

What would be the costs of implementing the information management system described in Scenario 1? Large direct costs for hardware and software would be incurred. Because the task of creating an integrated information system would be novel, some waste would occur, e.g., some inappropriate equipment probably would be purchased and programs would need to be revised to incorporate new features. The indirect costs might also be very high. Eventually, middle management and staff positions would be lost. Division officers might lose prestige and many of the training experiences they now have in decentralized organizations would not be possible. Much retraining of personnel at all levels would be necessary. Some coercion and turnover of employees would likely occur and the individuals involved might be harmed.

The benefits of Scenario 1 would be primarily indirect, intangible, and difficult to document. They could be derived from better and more timely information which would have great value in a turbulent environment. The group decision process and structured strategic planning in Scenario 1 may improve decisions and plans. Overall operating costs might be lowered and profits increased. Some individuals both inside and outside the company would benefit, particularly those associated with computerized information systems.

The same variety of hardware and software costs as those incurred under Scenario 1 would be expected for Scenario 2, but their magnitude would not be as great. In addition, there might be opportunity costs and threats to organizational sur[illegible] if other organizations realize Scenario 1 outcomes and benefits.

What would be the benefits of Scenario 2? The organization would gain any benefits of decentralized decision making in a structured hierarchy. Managers would have more information and possibly more control over operations. The evolutionary nature of the changes and the slower pace of change would probably be easier for many organization members to accept. Top managers would not have to work with computers, familiar media would be used to transmit information. The organization may have a more "human" climate. Profits might be improved as a result.

Summary and Conclusions

The analysis of the costs and benefits does not clearly favor either scenario, and many managers might conclude that they should position their organizations somewhere between these two. Such an approach toward implementing information management may actually be worse than moving toward the extremes. The organization might incur significantly higher costs without necessarily realizing the advantages of either scenario. On the other hand, a middle course may give the organization the flexibility and experience needed in the 1990s to move in the direction that is ultimately proven to be most successful. Organizations with similar products and services in their divisions may find greater benefits under Scenario 1 since an integrated information management system would be easier to design and implement. But a conglomerate in which managers at the strategic level avoid direct involvement in operations might be better off with a decentralized system like the one described in Scenario 2.

Thus, scenarios such as these do not lead to clear cut solutions. Rather they may serve to identify and to refine issues that must be dealt with, for instance, is designation of a strong data administrator the major factor influencing the outcome in Scenario 1? Is the change to group decision making necessary or desirable? What other scenarios are plausible? What is the most likely one? The future will always be unknown, but thinking about alternative futures and the issues that they raise may increase the likelihood that a desirable future will be realized

References

[1] Ackoff, R.L. "Management Misinformation Systems," *Management Science*, Volume 14, Number 4, 1967, pp B-145-B-156.

[2] Anthony, R.N. *Planning and Control Systems: A Framework for Analysis*, Graduate School of Business Administration, Harvard University, Boston, Massachusetts, 1965.

[3] Bass, B.M. *Organizational Decision Making*, Richard D. Irwin, Inc., Homewood, Illinois, 1983

[4] Blumenthal, S.C. "Breaking the Chain of Command," in *Management Systems*, Schoderbeck, P., ed., John Wiley and Sons, New York, New York, 1967, pp. 102-107.

[5] Curtice, R.M. "Data Independence in Data Base Systems," *Datamation*, Volume 21, Number 4, April 1975, pp. 65-71

[6] Davis, G.B. *Management Information Systems. Conceptual Foundations, Structure and Development*, McGraw-Hill Book Company, New York, New York, 1974.

[7] Dearden, J. "Myth of Real-Time Management Information," *Harvard Business Review*, Volume 44, Number 3, May-June 1966, pp. 123-132.

[8] Ein-Dor, P. and Segev, E. "Organizational Context and MIS Structure: Some Empirical Evidence," *MIS Quarterly*, Volume 6, Number 3, September 1982, pp. 55-67

[9] Hirschhorn, L. "Scenario Writing: A Developmental Approach," *American Planning Association Journal*, Volume 46, Number 2, April 1980, pp. 172-183.

[10] Huhn, G.E. "The Data Base in a Critical On-Line Business Environment," *Datamation*, Volume 20, Number 9, September 1974, pp. 52-56.

[11] Inbar, M. *Routine Decision Making: The Future of Bureaucracy*, Sage Publications, Beverly Hills, California, 1979

[12] Kahn, H. and Weiner, A.J. *The Year 2000*, Macmillan, New York, New York, 1967.

[13] Leavitt, H.J. and Whisler, T.L. "Management in the 1980's," *Harvard Business Review*, Volume 36, Number 6, November-December 1958, pp. 41-48.

[14] Lewis, C.J. "Understanding Database and Data Base," *Journal of Systems Management*, Volume 28, Number 9, 1977, pp. 36-42.

[15] Martin, J. *Computer Data-Base Organization*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1977

[16] Mintzberg, H. *The Structuring of Organizations*, Prentice-Hall Inc., Englewood, New Jersey, 1979.

[17] Mintzberg, H., Raisinghani, D., and Theoret, A. "The Structure of 'Unstructured' Decision Processes," *Administrative Science Quarterly*, Volume 21, Number 2, June 1976, pp. 246-275

[18] Nolan, R.L. "Computer Data Bases: The Future is Now," *Harvard Business Review*, Volume 51, Number 5, September-October 1973, pp. 98-114.

[19] Reside, K.D. and Seiter, T.J. "The Evolution of an Integrated Data Base," *Datamation*, Volume 20, Number 9, September 1974, pp. 57-60

[20] Robinson, S.L. "Computer Data Bases: The Future is Tomorrow," *Computerworld*, Volume 12, Number 38, September 1978a, pp. 31-32.

[21] Robinson, S.L. "'Future Shock' Seen Coming in Data Base Use," *Computerworld*, Volume 12, Number 42, October 16, 1978b, pp. 36, 38.

[22] Rockart, J.F., Ball, L., and Bullen, C.V. "Future Role of the Information Systems Executive," *MIS Quarterly*, Special Issue 1982, pp. 1-14.

About the Author

***Daniel J. Power** (Ph.D., University of Wisconsin-Madison) is an Assistant Professor of Management at the University of Maryland. Power is a coordinator of the Strategy and Planning Research Group and he is affiliated with the Center for Innovation and the Center for Automation Research at the University of Maryland. His research interests include corporate acquisition decision processes, computerized management decision aids, forecasting and planning tools, and individual decision behavior.*

CLUSTER III PAPERS

SECURE MULTI-MEDIA TELECONFERENCING

A STUDY FOR

SOFTWARE ARCHITECTURE & ENGINEERING, INC.

Sigma Associates

September 15, 1983

SECURE MULTI-MEDIA CONFERENCING

OVERVIEW

Teleconferencing - two or more locations communicating via electronic and/or image producing facilities - runs a spectrum from the simplest audio teleconference to the function-rich interactive motion video systems. Multi-media systems integrate voice, video, facsimile and computer-interactive modes of communication. This study will focus on two forms of teleconferencing, specifically that of audio teleconferencing and motion interactive video teleconferencing.

Virtually all audio conferencing is done over voice grade (2700 Hz) transmission facilities. Video conferencing, however, requires high bandwidth (up to 6 Mhz). This study will focus on the equipment and systems associated with these teleconferencing systems, will specify the transmission requirements, but will fundamentally assume that those required transmission facilities will be provided by DCA. The equipment discussed is that which is basically considered to be off-the-shelf in the mid term time period (1989).

For the purposes of this report, the discussions of audio conferencing and motion video conferencing will be treated separately, each with its own product description, functional requirements, and operational capabilities.

AUDIO CONFERENCING

Audio conferencing is nearly as old as the telephone and is today the most widely used form of teleconferencing. Audio conferences account for more than 90% of the conferences held via electronic media; anytime three or more people at two or more locations confer over the telephone, an audio conference is held. Even though the telephone was developed basically as a two-party communications device, not intended for group communications, individuals can still participate in an audio conference through an ordinary telephone. More elaborate conferences involving groups gathered in conference rooms or conferences between multiple locations require specialized equipment considerably more complex than the basic telephone.

This report will examine the full range of audio conferencing equipment and services in general use today, including the very basic three-party connections made over standard telephone instruments to the more sophisticated multi-party multi-location conferences using state-of-the-art equipment and specially designed conference facilities. For the purposes of this report, the following categories of audio conferencing apply:

- Conference Call - Basic. This type of audio conference is made among three or more parties using a standard telephone over the Public Switched Network (PSN) using the conferencing capabilities of the telephone instrument, PBX, Centrex, or telephone company conference operator. No specialized equipment is required.

- Conference Call - Enhanced. This level of audio conference is set up in precisely the same manner as the basic conference call between two or more locations with the exception that specialized equipment is used at the conference locations to allow a "hands-free" operation or to permit more than one person to participate in the conference at a given location. This specialized equipment is nothing more than a microphone (or multiple microphones) used for voice pickup and a loudspeaker to amplify and broadcast the incoming speech.
- Group Conferencing. A conference held between two or more locations, with at least one location being a conference room designed to accommodate several conferees is termed a Group Conference. Highly specialized equipment is required to maintain audio quality; in addition, stringent design considerations apply to the development of the conference room to avoid or eliminate undesirable acoustical properties.
- Audio-Graphic Conference. This rapidly growing category of audio conferencing involves the use of visual information to enhance the audio conference. Electronic blackboards, facsimile machines, and slow scan video units are used to transmit graphics between conference locations. A highly advanced form of audio-graphic conferencing is audio-graphic with database augmentation, whereby data is actually manipulated before transmission or projection. Data base augmentation provides a means to play out "what-if" scenarios with the presentation material.

Any of the above categories of audio conferencing can become a subset of any other category. For example, a group audio conference with graphic augmentation can involve participant locations using basic or enhanced conference capabilities. All categories of audio conferencing, including audio graphic with slow-scan video, can use standard voice grade facilities on the Public Switched Network or can use private line facilities, either switched or non-switched. The use of wider (than voice grade) bandwidth is not only unnecessary, but also undesirable as the added frequency response can over-emphasize the poor acoustical properties of most conference facilities.

Examples of each category of audio conferencing cited in this section are described following:

<u>Conference Call - Basic</u>

The most rudimentary form of teleconferencing is also the most widely used due to its pure simplicity and ease of use. To initiate a basic conference call, a user simpiy adds many conferees to the telephone connection as is desired or technically possible through one of the following methods:

- o Instrument conferences, where multiple lines can be accessed from within the telephone set.
- o System conference, where the telephone system (key, PBX, or Centrex) sets up the conference on commands issued by the user from the telephone set.
- o Operator conference, where the telephone system or telephone company sets up the conference.

Typically, a basic conference call is limited to between three and six conferees depending on the capabilities of the particular system used to set up the conference. Audio quality rapidly degrades beyond a six-party conference due to distortion caused by signal attenuation noise accumulation, and circuit imbalance. A basic conference call is further limited in that only one person per telephone set location can participate in the conference.

Conference Call - Enhanced

This improved form of audio conference adds microphones and loudspeakers to the conference connection, thus allowing several persons in the same general area as the conference equipment to participate in the conference. The most common example of enhanced conference equipment is the Bell 4A Speakerphone, although there are many similar units on the market from a variety of manufacturers. In addition, many portable units are available that can be easily carried to convenient sites, thereby greatly increasing the utility of this type of conference. Call set up for the enhanced conference call is exactly as in the basic conference, and has the same limitations on the number of conference connections.

The enhanced conference call has an additional limitation in the number of persons that can use a single speakerphone. Generally, the speaker should be within two or three feet of the voice pickup; a "rainbarrel" effect becomes quite pronounced as the distance from speaker to microphone is increased. To eliminate feedback, singing, and far-end talker echo inherent in a two wire Public Switched Network connection, many speakerphone systems use a voice-switched gate, allowing only one conference location to speak at once, effectively making the circuit half duplex. In early conferencing systems, interrupting the speaker was difficult because as long as the speaker was talking, the speaker's receiver was cut off. If someone wished to interrupt the speaker, that person would have to wait for a pause in the speaking in order to interrupt. State-of-the-art equipment available today uses logic gates that can quickly switch back and forth between transmit and receive paths to allow a more natural conversation to take place.

Group Conferencing

A teleconference involving from 6 to 30 people in medium to large-sized conference rooms at two or more locations fits this category of audio conference. Whenever conferences extend beyond the private office and into "conference rooms", special design practices must be employed for both the conference room equipment and the conference room. This type of conference setup generally use multiple microphones of a significantly higher quality than those used in small conference setups. By using an array of microphones strategically located throughout the conference room, each speaker is assured of being within a reasonable distance from the voice pickup. The number of microphones and the location of speaker and microphone can be optimized for the characteristics of the particular conference room. Some systems cluster the microphones about a vertical line central to the majority of speakers. These types of microphones provide satisfactory voice pickup at distances up to twelve feet from the speaker; high power amplifiers drive speakers placed on the conference table or about the room, or mounted in the ceiling.

The most sophisticated conference room equipment developed specifically for group conferences is the "conference bridge" used to interconnect the transmission circuits from each conference location. The most advanced of these conference bridges are "active" bridges in that the bridge automatically compensates for circuit imbalance and loss on each line of conference. Circuit noise that limits most audio conferences to six connections is minimized in a conference bridge by the use of a voice-switched circuit which attenuates the receive path (to the bridge) of an inactive (non-talking) connection. Similarly, at the conference bridge location, the transmit path is cut off when no one is talking in order to prevent conference room background noise from being introduced to the connection. State-of-the-art conference bridges used today can accommodate up to 28 connections.

Call set-up procedures for group conferences are identical to those used for basic and enhanced conference calls; group conferences using conference bridges often use a conference attendant to establish the conference connections. In addition to attendant operation, conference bridges are usually equipped with a "Meet-Me" feature allowing individual conferees to dial into the conference at a predetermined time. The conference bridge will answer the call and add the caller to the conference connection, usually with a warning tone to alert the participants to the presence of another conferee.

Where basic and enhanced audio conference arrangements were quite easy to implement, group conference arrangements require a significantly greater level of effort to implement in order to maintain acceptable audio quality. At a minimum, group conferences imply the use of a conference facility specifically designed for teleconferencing. In a dedicated conference room, the conferencing equipment is usually installed out of the view of the conferees. All equipment is fine-tuned to meet the particular operating characteristics of the conference room; often, some acoustical treatment is applied to the room to improve the acoustical properties of the room. Ideally, the teleconferencing room should be located in the building interior where it is relatively isolated from outside noises but should not be located near the core where building equipment such as elevators, HVAC systems, and plumbing could cause electrical or audible conference interference.

The conference room should be sized primarily to meet the needs of the conference participants; a medium-sized conference room of 350 square feet will accomodate fifteen conferees comfortably. The dimensional proportions of the conference room and the makeup and placement of the materials used in the conference room must be chosen with careful regard to standard acoustic engineering practices. As the room size grows, so do the acoustical problems; these problems can be maintained at an acceptable level if the conference groups are kept at a reasonable size and if considerable care is taken in the design of the room.

<u>Audio-Graphic Conferencing</u>

This category of audio conferencing is the newest and most rapidly growing form of teleconferencing. Basically an enhancement to the group conference, audio-graphic conferencing uses visual information to enhance the conference. This visual information is distinguished by the type of graphics that can be transmitted and are categorized as follows:

- o Facsimile. The transmission of previously prepared documents, whether typewritten or drawn, is known as facsimile. Anything that can be shown on a sheet of paper can be transmitted via facsimile; this information can be used to greatly enhance the utility of an audio conference.
- o Telewriting. Telewriting is the instantaneous transmission of hand-drawn information, such as graphics, figures, sketches, etc. The equipment at the transmitting end typically resembles a stylus or a blackboard that is specially designed to convert hand-drawn text to a signal suitable for transmission over ordinary telephone lines. The receiving end uses a video monitor to display the transmitted text.
- o Slow-scan video. Any image that can be picked up by television camera can be transmitted over telephone lines to a distant video monitor. People, graphics, engineering drawings, etc., can all be used to enhance a video conference. At the receiving end a still image is presented that is reconstructed approximately every 30 seconds.

All audio-graphic conferencing can be done over ordinary voice-grade telephone lines using either the Public Switched Network or private line facilities. Call set up for the audio portion of the conference is exactly as in the other categories of audio conferencing. The graphics portion of the conference is set up according to the unique characteristics of the terminal device. For example, a slow-scan video camera can be "conferenced" to multiple video receivers in the same manner as the audio connection, including the use of a conference bridge. However, the camera is often manipulated during the video transmission to, for example, focus on a particular person or object or to provide a panoramic view of the conference table. Facsimile machines have their own individual operating characteristics as do telewriters.

Since audio-graphic conferencing is usually an enhancement to group conferencing, it requires the highest level of effort to implement. All of the effort needed to implement a group conference is required; in addition, extra connections must be established for the graphic equipment and provisions must be made for the placement of this equipment.

If data base augmentation is to be included in an audio-graphic conference, then some means of data base access and data manipulation must be provided. A mini-computer at the conference control location is usually the means for both data base access and data manipulation.

Discussion of Functional Requirements

The functional requirements for audio conferencing range from the need to add input from a third party for decision making to the "shirt sleeve" operating meeting described in the video conferencing section of this report. Following are some areas of the major functional requirements of audio conferencing:

- Ad-hoc conferencing - The spontaneous need to add one or more additional parties to a two party conversation to cousult, confer, or inform.
- Planning meetings - A form of project management conducted over the telephone to kick-off a new project, review existing project status, or to develop strategies and action plans.
- Task force meetings - Basically a subset of project management, task force meetings can take place through audio conferencing. With the addition of graphics, this type of meeting can be highly productive and can minimize the expenses and non-productive associated with travel to task force meetings.
- Education and training - One of the most widely used applications for audio conferencing, education and training programs can be effectively carried out through any level of audio conferencing other than the basic three party conference call. A lecture can be extended to a remote location via two way audio or can be enhanced with "electronic blackboards" and slow scan video.

Options/Levels/Variants

All of the basic levels of audio conferencing have been described in the previous section; however, the single most significant option is an audio conferencing system is that of transmission facility selection. Unlike video conferencing which requires dedicated facilities associated with a single network, audio conferencing has a number of variants. The transmission facility options are as follows:

- o Public Switched Networks
- o Private Switched Networks
- o Dedicated Private Line, Non-Switched
- o Dedicated Private Line, Switched

The most convenient and universal method of conducting an audio conference is over the Public Switched Network. Literally any public telephone in the world can be used as a conference terminal. Because the network is two-wire at the terminal ends, circuit balance must be carefully maintained to avoid singing, echo, and other forms of feedback distortion.

Private Switched Networks such as CCSA, EPSCS, ETN, and AUTOVON offer yet another option for the audio conference. Further, many of these private networks have the capability to interconnect with the Public Switched Network, allowing a greater degree of flexibility to the conference. Some private switched networks such as EPSCS and AUTOVON are four-wire end-to-end, avoiding some of the balance and speaker protocol problems associated with two-wire connections.

Non-switched dedicated private lines are used between conference center locations where the traffic volumes are sufficiently high as to justify the use of a dedicated facility, and where the conference points are predetermined and static. The most basic of these would be a two-point voice grade private line between two conference locations. Multi-point dedicated private lines used for audio conferencing exist today between as many as 150 locations, each location equipped with its own "push-to-talk" handset and loudspeaker. The conference network is basically an "open-line", with speaker protocol being the only determinant as to who is allowed to speak and when. Dedicated private lines can be configured as four-wire end-to-end to improve transmission characteristics.

Switched dedicated private lines are often used for audio conference networks where traffic volume is high and where the conference locations are fixed, but vary from conference to conference. As many as 100 locations can be on the network, but conferences are usually limited to about six locations. Anyone wishing to initiate a conference simply checks the line for availability then dials the station code(s) of the locations to be conferenced.

Any of the four transmission facility options can be used with any category of audio conference; moreover, transmission facility options can be intermixed among themselves and among intermixed categories of audio conferencing. For example, a group conference call can be established using the facilities of both the Public Switched Network and a private network with access to the public network. As a conference call becomes more complex in its architecture, maintaining quality becomes more of a problem.

Private line facilities offer a somewhat higher level of security and survivability than does the public network. Private lines can be routed over specially designated facilities to, for example, avoid certain routes or types of facilities such as microwave and satellite. Private line facilities generally perform at a higher level than public facilities; they can literally be "fined-tuned" to meet the particular requirements of the conference network.

Technical Challenges

Basic audio conferencing is a mature technology and as such has been fairly well developed; the primary technical challenges for audio conferencing lie in the areas of improved conference bridges and enhanced graphics capabilities. Voice compression is not an issue when voice is transmitted over analog facilities; however, as digital transmission becomes more common, it will be necessary to further reduce the digital bandwidth required for accurate voice reproduction. Current encoding schemes allow adequate quality voice to be transmitted at 32 Kbps (for comparison, an ordinary high quality voice grade facility has a maximum transmission rate of 9.6 Kbps). More advanced encoding techniques promise to further reduce the digital bandwidth required. This is perhaps the greatest technical challenge for audio conferencing: the digital transmission of voice at voice grade data rates.

Capabilities of Operational Audio-Conference Systems

The University of Wisconsin-Extension: Educational Telephone Network.

The University of Wisconsin-Extension is perhaps one of the most prolific users and promoters of audio conferencing. Over 200 sites in the state of Wisconsin are equipped with portable audio conferencing units. Course instructors can remain on campus or can conduct classes from virtually any convenient location; students attend class at the nearest public facility equipped with the conferencing unit.

NASA Audio Conferencing System.

The initial requirement for the NASA system grew out of the space program when geographically dispersed contractors and NASA locations had a need to communicate in working sessions on the Apollo project. Many locations were equipped with telecopiers for the tranmsission of printed and graphic material. The NASA audio conferencing system is still being regularly used for committee meetings and for project coordination.

Human Services Development Institute: "Northern Network".

The Human Services Development Institute at the University of Southern Maine in cooperation with the Maine Bureau of Rehabilitation established a switched dedicated private line audio conference network serving twenty vocational rehabilitation offices in Maine and New Hampshire. The conference network was set up to provide rehabilitation services to handicapped clients; the impetus for the network was to avoid the difficult travel in the rural areas of Maine and New Hampshire. The experimental program proved to be highly successful and is still in continuous use today.

Capabilities of Subsystems

The major subsystems of a fully developed audio-grpahic conferencing system are:

- o Telewriters
- o Facsimile machines
- o Slow-scan video units

The capabilities of slow scan video units are covered in the video conferencing section of this report; following is a discussion of telewriter and facsimile machines.

Telewriting is the instantaneous transmission of hand-drawn graphics such as pie charts, circuit schematics, mathematical equations, and chemical formulas. These images are transmitted to the receiving end over ordinary voice grade telephone facilities, either public or private, switched or non-switched. At the receiving end the graphics are displayed on video monitors or on hard copy. The graphic information is produced by "writing" on an electrically sensitive surface which converts the hand drawn text into a digital signal, which is then encoded into a format suitable for transmission over the appropriate facility.

The earliest form of telewriting were the graphics tablets used in department stores, warehouses, and parts depots by order takers to enter a customer order onto a tablet and simultaneously transmit the written order to the order filler. Possibly the most widely known telewriter today is Bell's *Gemini 100 Electronic Blackboard*. This "blackboard" allows a speaker to write, using ordinary dustless chalk, on a "normal" blackboard surface. Each point on the blackboard surface is represented by x and y coordinates; by touching the surface with the chalks an encoder sends the x and y coordinates of that point to the remote unit. The remote unit encodes the information and illuminates the appropriate points or lines on the TV monitor that correspond to the hand-drawn graphic.

Facsimile systems are used to relay virtually anything that can be shown on a single piece of paper to distant sites over ordinary telephone facilities or private lines. The printed information is converted into electric signals through a scanner. This signal is then transmitted to the distant end and reconstructed into a printed page.

Facsimile machines are classed into the following categories:

- Group I - Facsimile machine which operate at four to six minutes/ page and use FM analog modulation.
- Group II - Facsimile machines which operate at one two to three minutes/page using AM analog modulation.
- Group III - Facsimile machines which operate at one minute or less per page and use digital techniques to enhance the speed.
- Group IV - Facsimile machines which are classified as high speed digital and high resolution, transmitting at approximately 56 Kbps.

For audio graphic conferencing over standard voice grade facilities, only Groups I, II, and III devices are used; however, full-motion video teleconferencing facilities are often designed with the capability for Group IV facsimile.

Five Year Forecast

The forecast for audio conferencing system is much the same as it has been for the past two decades; equipment and facilities will continue to be readily available for all levels of audio conferencing for the forseeable future.

Analysis

The key system elements for audio conferencing are:

- Transmission facilities
- Audio terminals
- Bridging devices
- Graphic terminals

Transmission facilities are as easy to obtain as telephone service and cost the same. Private line facilities can be obtained in 18 working days from most common carriers; AT&T, is normally used as the benchmark for price comparisons. A 500 mile voice grade private line from AT&T is $700.00/month between two points. Costs for a multi-point line switched network vary widely depending on level of sophistication required and geographic dispersion. Lead time to procure a large multi-point network could be as much as six to nine months. Following are some purchase prices for other major system components:

Audio terminals	$300 - $1,800
Bridging devices	$1,200 - $22,000
Facsimile machines	
Analog	$3,000 - $5,000
Digital	$6,000 - $19,000
Electronic Blackboard	$9,000 - $12,000
Video Camera	$1,100 - $2,000
Baseband monitor	$600 - $800

MOTION INTERACTIVE VIDEO TELECONFERENCING

Until the last few years, the idea of using electronic means of conducting business meetings such as the recurring shirt sleeve sessions in which managers and professionals spend so much of the business day, was not generally accepted and implemented except in a very few organizations. The ability to save travel expense and the time of the individuals involved in that travel for the purposes of a meeting, were not sufficient to offset the break in practice and in culture that resulted. However, in the last several years, many more organizations have begun to install and utilize teleconferencing systems since the experience of those initial pioneers has proven that there are many benefits to such a system, all summarized in that generic category of increased personal productivity, but resulting from:

o Shorter meetings

o Better preparation

o Faster decisions

o Better cooperation

o Greater meeting effectiveness

A number of surveys of business users, including that by Hansel & Green of Satellite Business Systems, support these results. The organizations surveyed cited benefits not only from the productivity of their individual members, but from a more decisive environment, one based on more and better communications, and with such overall results as faster introduction of new products, quicker reaction to an unplanned event, better decissions, and the overall effectiveness of the organization.

Conclusion

The multi-media motion video teleconferencing system consists of a carefully designed meeting room, the various cameras, projectors, and viewers of the information both at the local and remote sites, a high quality audio system, the communicatons interfaces including the codec, a control console to manage the various subsystems, the means to transmit data or copies of material in addition to that which is being communicated via video, and the transmission media. This study addresses those elements of the system up to and including the interface with the transmission media.

The current utilization of these systems is limited to eight or nine large commercial organizations and two major communications vendors. The experience is fairly recent, starting in the early 1980's, but has been proven extremely beneficial for those using this system. There is currently underway a major increase in the number of organizations planning to implement such systems.

The typical performance expected of these systems is one of providing for a normal shirt sleeve operating meeting environment typical of that conducted by the organization involved, but with the meeting participants divided among two or more rooms geographically separated. All equipment must be able to support a real time interactive environment. The system operation must be non-obtrusive; the operation must be natural and not studio-like so as not to detract from the effectiveness of this communications medium.

While there are a wide range of alternatives to the design of the rooms involved and the equipment being employed there is a minimum basic complement of equipment which currently costs approximately $200,000-$300,000 per room. The room construction, of course, is a function of the type of meeting conducted, and the level of comfort and atmosphere desired by the using organization. The implementation of a teleconferencing network requires a minimum of 12 to 15 months for a reasonablly paced program of room and system design and implementation and establishment of the associated transmission network. Some level of dedicated staffing is required at each room location to ensure the proper maintenance and operation of the conference room; vendors will provide maintenance support. There is also the requirement for some central network control system with the associated equipment and staffing. The in-house resources to implement such a system are not estimated here but are a function of the specific process within the user organization. Timelines for decision making relative to the implementation activities are also not estimated since this again is a function of the process of the organization involved and the priority associated with this program.

Discussion of Functional Requirements

In the most general sense, the overall functional requirement is that of creating a typical shirt sleeve operating meeting environment currently being utilized by the organization involved. For practical purposes this generally involves a meeting of six to eight participants (those seated at the conference table and actively participating in the meeting). With a similar number at the remote location, this provides the ability to have a meeting of 14 to 16 active participants, generally at the limit for a reasonable meeting. In addition, there can be other support personnel present in the meeting rooms who are available to provide information or views.

Other major considerations involve the ability to provide:

- o A view of the speaker or presentor.
- o A view of the other participating meeting room(s), showing all participants.
- o The ability to focus on the individual speaker at the conference table.
- o Other normal meeting support typical of that organization including slides, vu-graphs, flip charts, and blackboards.
- o A means to obtain and interact with ad hoc material such as back-up presentation material.
- o The ability to interact with the presented material (i.e., marking up a vu-graph, creating a hand written chart on a flip chart or blackboard, etc.).
- o Access to background data in a local or remote data base.
- o The ability to access other staff personnel not in the meeting room.

The various pictures of participants and information being presented and discussed must be clear and large enough to be easily read. Where motion is involved, it must be natural and without blur. Color is important, particularly for pictures of the presentor and the meeting participants, to provide a natural environment. Voice must be of a high fidelity and in sync with the picture. Any support material being developed and introduced into the meeting must be available with a 10 to 20 second response, which is the time that would normally be required to change a slide in a meeting, to look for a back-up piece of information, etc. The room supporting equipment and the operations must not be studio-like and the operation of the various elements of the equipment must be non-obtrusive. The control system must be extremely easy to learn and to use; it must not detract from the substantive aspects of the meeting itself. Provisions should be made for the presentor to also view, via monoitor, the participants in the distant location so that eye-to-eye contact can be maintained by the presentor who is generally standing at the front of the meeting room.

In addition to these general requirements, the WWMCCS environment also dictates security and survivability considerations. A teleconferencing room can be developed in a secure mode. The transmission can and should be encrypted. Relative to survivability, normal requirements existing for the application involved for the survivability of room and equipment can be factored in. Survivability of the transmission network can be considered via redundant transmission paths, but this can be extremely difficult and expensive with the high bandwidths involved in a motion video teleconferencing system.

The ability to operate in a degraded mode would be a dramatic drop back to a voice grade type of operation. With careful planning, this degraded mode can, in essence, become that of the audio graphic teleconferencing approach described earlier in this report.

Options/Levels/Variants

There are several opportunities to reduce the level of functions described above with the attendant reduction in equipment costs as well as the extent and cost of transmission capacity. These involve operating at less than the current 1.544 Mbps required for transmission of a quality full motion image. For instance, it is possible to operate at 896 or 768 Kbps which provides a color image but which introduces blurring where there is fast motion by the individual such as walking, arm movement, etc. There is also the opportunity to operate in black and white, but again this would detract from the basic premise of a "natural setting". Audio can be less than high fidelity and transmitted via normal voice grade lines, but would be subject to a degradation in the voice aspect of the teleconference (the true cornerstone of any teleconferencing system is the audio portion). In all cases, the above would be somewhat counter to the investment committed to for a motion video teleconference, which implies a commitment to a real time on-site natural meeting environment.

There are a number of alternatives to augment the basic requirements discussed, including:

o An additional TV camera, the associated monitor, and the additional transmission capacity can be added to provide a "continuous presence" aspect of the teleconferencing system. AETNA has included such a capability in their system and finds that it is highly desirable in maintaining the natural meeting environment. The extra camera, the extra equipment for monitoring and the extra high bandwidth transmission path are the price for this enhancement. However, an approach has recently been developed which involves the transmission of only the middle portion of each picture (the motion part). If effective, this would hold the transmission bandwidth requirement to that of a single video channel.

- o Stereo voice can be added to improve the sound quality as well provide a spatial presence. Equipment and additional transmission capacity are the considerations.
- o The facsimile unit and overhead graphics camera considered basic to the system can be replaced with a high resolution scanning and display system. This system is assembled of hardware from high speed facsimile and high resolution video technologies operating at speeds approximating 448 Kbps. They can provide a new image within five seconds at the remote location, operating in monochrome. This approach to improved graphics has been used in very few systems. An alternative has been recently made available in the form of a codec feature which allows graphic transmission on the video channel during a short video interrupt period.
- o Rather than using TV monitors (19 or 25 inch), a large display screen (4x4 feet or larger) with the equipment for expanding the image can be added. This is a positive enhancement where utilized today. High resolution screens using up to 1024 scanning lines provide higher picture quality, with the attendant higher costs.
- o The integration of an access to local and remote data bases with graphics generation and display equipment can be added.

- Interactive graphics such as an interactive over head graphics package, can provide additional natural meeting room capabilities.
- A codec to operate at still frame operation (56 Kps) can also be added to provide operations in a degraded mode if the high bandwidth links are lost. This would also support applications which do not require the full motion capability and as such the transmission costs for those meetings can be reduced.
- In addition to the point-to-point, two room type of operation described above, a multiple location teleconferencing system can be implemented with its attendant complexity and cost. The full interactive multipoint-to-multipoint type of video teleconference requires a replication of equipment and a significant and possibly unacceptable amount of transmission capacity. There is a requirement to develop the software necessary to operate the meeting on a decentralized control basis. This has been estimated at $750,000 to $1,000,000. In a less structured mode, a three way conference has been run in the AT&T PMS environment. This requires a replication of equipment and multiple transmission paths. One alternative which has been investigated is the use of a point-to-"shifting point" type of configuration where at any point in time one location is broadcast to all others, but only two of the locations are interacting in a video mode. The pair of interacting nodes can be changed during the teleconference. This requires the use of a satellite transmission facility which has the ability to re-[illegible]ate bandwidth among the nodes.

The WWMCCS environment also requires international communications. With the high bandwidths involved, this may involve a double hop via satellite. This is achieveable technically without degradation to the image, but requires the tolerence with the approximate one second delay for the double round trip via satellite. This has been set up and demonstrated by Satellite Business Systems in San Francisco to Chicago to London teleconference. A similar hook up to Tokyo is planned later this year. It is found that with the simple acceptance of "be polite and wait" - twice as long as you are accustomed to waiting on an international phone call - the meeting can be held satisfactorily with all of the benefits of the full motion video conference.

Performance

In summary, the performance of a motion video conferencing system requires a high fidelity full motion camera and monitor system. While the normal 525 line TV monitor is appropriate for the people images, quality graphics dictate a higher resolution transmission display system. Allstate, in their video conferencing system, utilize a Rapicom developed system operating at 1024 lines, projected on to a 5 foot screen.

Audio must be clear and two way such that participants at both locations can both speak and hear simultaneously so that speaker interruption can occur naturally. Conferees must be able to walk around the room and hear and be heard from anywhere within that room.

The setting of the room and the operation of the equipment and the control console must be unobtrusive and easy to learn and operate.

Where new information is being introduced in the form of presentation material, it must be available in a normal time span of 10 to 20 seconds.

Technical Challenges

There are four primary areas of technical challenge. The first is that of the room design and equipment selection. Experience is being gained daily with the existing systems which today are basically limited to those operational systems utilized by Aetna, Allstate, American General, Citicorp, DEC, Arco, ISA, Liberty Mutual, Procter and Gamble and AT&T/PMS. Selection of the proper equipment and, more importantly, the integration of this equipment into a system is critical.

The second area of technical challenge is to continue the current trend in codecs of reducing the transmission speed required to transmit a high quality video image. This capability was only reduced from 3 Mbps to 1.5 Mbps within the last several years. Quality video transmission @ 896 and 768 Kbps is just being introduced. R&D continues in this area and will result in continued improvements over the next five years.

The third area of technical challenge involves that of effecting some level of standards for transmission interface and codec speeds. While work is currently being done to reduce the transmission speeds of the full motion video codecs, the vendors in general will offer their equipments at non-standard speeds such as 448, 512, 768, or 896 Kbps. Standardization among rooms, and the ability to interface with standard transmission media will be a challenge.

The last challenge is that associated with the control system. The teleconferencing rooms today, both full motion and freeze frame type of operations, require the utilization of some sort of control system and panel. Much progress has been made in this system, but there is still development work required to improve the ability of the system to be truly nonobtrusive and extremely simple to learn and operate.

Capabilities of Operational Systems and Subsystems

The following discussion of operating systems is a composite of those teleconferencing systems currently in operation. The room and system designs generally reflect the feedback from extensive trials by a number of these organizations. As an example, Aetna Life Insurance started with a pilot test involving a four room configuration in two locations, ten miles apart, in Hartford and Windsor Locks CT. During the two year period of test and experimentation from March of 1981 through February 1982, approximately 30,000 people utilized this full motion teleconferencing capability in approximately 5,000 different meetings.

The major components of a full motion viedo teleconferencing facility are as follows (all equipment is not required; use of some mutually excludes others):

o Participant camera - the images of participants are generated by a color camera at the front of the room. This camera is mounted on a motor driven platform and is equipped with a zoom lens. When a button on the control console (corresponding to a seat position) is depressed, the camera will automatically rotate and zoom to a preprogrammed position by command from the control console or by audio pickup (a very difficult problem). There are six to eight of these preprogrammed positions which can be easily changed.

o Camera for the presentor - this is comparable to the participant camera but does not require the programmable pan.

o Remote participant monitor - a standard TV monitor (probably 25") which displays the video images (presentor, meeting participants, presentation material, etc.) being transmitted from the distant location.

- Remote participant display projector - a video projector located in the equipment room would display the distant conferees on a large (4 or 5' square) screen. This would allow all participants in the meeting to more clearly observe the received image and have a sense of a more natural environment than that provided by a TV monitor.
- Large presentation material such as flip charts - large charts (30"x40") could be displayed on a stand located at one corner of the room. Presentations could then be viewed by all meeting room participants. The camera for the presentor, located at the opposite corner of the room with a motorized platform and lens controlled by a local conferee, would pick up flip chart and presentor.
- Transparencies or slide material - page size transparent foils prepared for an overhead projector can be displayed both locally and picked up via the presentor and presentation material camera for transmission and displayed at the distant location.
- A video monitor displaying the distant conference room could be colocated with this camera to assist the presentor in maintaining eye contact with the distant conferees.
- Facsimile system - page size hand outs can be transmitted to the distant conference room with a high quality, medium speed (20-60 seconds per page) facsimile machine located in the equipment room.

- Audio system - a high quality audio system is required with the speakers and microphones being the only visible portions of the system. A significant amount of equipment to provide mixing, switching, testing, cancelling, amplification, and the communications interface is contained in a nearby equipment room. This system is full duplex allowing both parties to speak and hear simultaneously. Sensitive microphones and high acoustical room treatment allows participants to move about the room when speaking and clearly be heard at the distant location.
- Transmission and interfaces - the primary communications interface unit is a video motion coder/decoder (codec). The basis components of the codec are a coder for outbound signals, a decoder for inbound signals and a power supply. In terms of video conferencing, the codec accepts a user's normal television signals in analog form and codes them into a digital signal for transmission. This is currently, in most installations, done at 1.544 Mbps. This codec can simultaneously receive a digital data signal from the remote site and decode it into an analog television signal to be seen at the local site during the teleconference. Encryption is available with these units.
- Audio system interface - the audio system interface and communications network connection could be included with the video codec for 2 point conferencing. However, multi-point conferences require separate codes and network connections since the audio and video must be independent. Mono or high quality audio requires a 56 Kbps network connection. A stereo high quality system would require a 112 Kpbs network connection. For added security, an encryption unit can also be added to this data connection.

- Conference controller - generally custom designed for the specific requirements of the using organization, a control console is currently required for operation of the various system elements, such as camera switching, zoom, and focus, seat selection for particant camera movement, audio volume adjustment, control of the hard copy functions, etc. Operational experience and the resultant improvements in room and system design are reducing the requirement for, or scope of, the controller.
- Computer for system control - unless included in the control console itself, a mini-computer or some other available computer capacity will be required to support the control console and testing functions. This computer capacity can also be provided by the computer associated with the network control function required for a multi-location network.
- Beyond the minimum requirement for a facsimile type of capability, many conferences require the distribution and display of hardcopy information such as memoranda, charts, graphics, drawings, statistical or financial information, etc. One approach is the high speed systems which are available which can scan, transmit, display, and reproduce a document in less than 10 seconds. Generally located as an inset to the conference table, these systems require a transmission bandwidth of 56 Kbps to as high as 448 Kbps. There is of course a tradeoff between response time and bandwidth for such a system. Recent advances in codec design also provide the capability for video graphics - transmitting this material over the video channel during a brief video interrupt.

- If a high quality graphic system is involved, there is generally a requirement to have a higher than normal quality display. In these systems a 1024 scan line screen and display provides the high visual clarity required for this type of information. Allstate uses such a system in its teleconferencing network.

Five Year Forecast

In general, all of these systems and subsystems required for a full motion video teleconferencing capability currently exist. In addition to the hardware and software involved, the emergence in the last few years of systems integrators and turn-key vendors for these systems is an important element of capability for the effective implementation of these systems.

The major areas of change in cost and/or performance over the next five years will be associated with the motion codec and the conference controller. Codecs will continue to be developed with the ability to transmit quality motion video at transmission rates lower than the 1.544 Mb being utilized today. Compression Labs and NEC are two of the primary vendors who are currently working on full motion codecs to operate at the 896, 768, and 512 Kbps transmission rates. Since these codecs are basically computer technology and software, their cost will decline due to the cost performance improvement trends for all digitally based products. Increasing volume of production will also reduce codec prices.

The requirement for the conference controller is being reduced as room and system designs are changed to reflect operational experience. Where still required, the conference controllers will also benefit from the cost performance improvements typical of digitally based products. More important, this element of a teleconferencing system will continue to

become easier to learn and operate in a non-obtrusive manner. Better human engineering, continued improvement in the understanding of the controller requirements
for a teleconferencing based meeting, and the introduction of the non-data processing type of features showing up in other office type equipment such as touch screens, use of icons, soft function keys, etc. will continue to improve these systems. Recent work by a number of vendors including M/A-COM and DDI have brought these types of features into the control system.

ANALYSIS

The current costs of the various elements of the system are summarized in Table 1.

The primary system elements which will decrease in cost in the five year period considered for this study are the motion codec, and the system controller. It is reasonable to expect these units to be priced at approximately 50% of their current levels five years from now. The higher speed, high resolution document or image transfer subsystems which have only recently been introduced, will also decrease in cost as production volume increases, but not by the same degree.

The implementation of a teleconferencing system requires approximately 12-18 months, in addition to that time required for the development of specifications and selection of vendors involved. The construction of the room and installation of the teleconferencing equipment can be shortened to 9-12 months, but the introduction of the high bandwidth transmission capacity requires a more reasonable planning horizon of 12-15 months.

TABLE 1

COST OF MOTION VIDEOCONFERENCING EQUIPMENT

Participant TV Camera	$20-$25,000
Presentation Camera	$10-$15,000
Audio	$20-$25,000
TV Graphics Camera	$5-$7,000
TV Monitors	$2-$4,000
Large Screen Projectors	$12-$15,000
Control Console	$20-$35,000
Switching & Test Equipment	$15-$20,000
Facsimile (20-60 Seconds)	$10-$15,000
High Resolution System	$80-$100,000
Motion Codec	$150-$180,000
Interactive Tablet	$30-$50,000
Minicomputer	$20-$40,000

1. Not all elements would be utilized in any one system or room.

2. A "basic package" would be approximately $200-$300,000.

3. There are other potential equipments which may be required to meet the specific user requirements.

Suggestions for a Development Plan

The recommended development plan is one that would be based upon a pilot program involving two sites. Such a system could be implemented within the continental United States with either satellite based or AT&T provided T1 transmission capacity. The recommended development plan is as follows:

- o A six month period for data gathering and analysis would involve discussions with those organizations such as Aetna, Allstate, Arco, ISA, and SBS on the development of their systems, the design of their rooms and their usage experience. Review of existing rooms and demonstrations can be arranged. In the case of Arco, which just recently started operations, this effort should include meetings with the critical organizations which supported the development of its implementation plan including the Annenberg School, Richard Burn Associates, and the Institute for the Future. Annenberg students had an extensive interview program with Arco managers and professionals. Burn Associates developed the implementation strategy including the publicity campaign and training within Arco. Institute for the Future worked on room design considerations and implementation issues. Discussions should also be held with organizations which are currently close to implementing major motion video conferencing systems. J.C. Penney and the May department stores are in their initial program development phase. Hercules is developing a plan to migrate from its existing system and extensive experience with slow scan teleconferencing to the implementation of a motion teleconferencing system. This phase would require 4-6 months.

- o The development of detailed requirements for the room and the operational systems and for development of the appropriate procurement documentation would then be undertaken. The requirements will generally focus in two areas, that of the room layout and the associated equipment, and the specific user peculiar requirements for the control console. A period of 3-4 months for this activity should be planned.
- o The selection of a systems integrator or turn-key vendor is key to the successful implementation of a video teleconferencing system. Rather than selection of individual equipment by the procuring agency (WWMCCS or its designated procurement authority), the key selection is that of the systems vendor. The selection of that vendor, the procurement of equipment, construction of the room, training of WWMCCS users, and the test and acceptance of the room and transmission systems would require 12-18 months.
- o A use and evaluation period is then recommended of approximately 9-12 months. The experience with the utilization of these two rooms in the WWMCCS environment and in support of the specific WWMCCS decision and meeting support process, would be the basis for modification if any to the previously defined system, and the basis for implementation of additional teleconferencing sites.
- o If the requirement were established for a multi-location interactive motion teleconference, a software development would have to be initiated. This can be undertaken in parallel with the above (with the attendant risk of a change requirement) or initiated after the start of the pilot program (with the attendant protraction in overall schedule).

CONCLUSIONS

In both of the teleconferencing scenarios discussed in this study, the various equipments are available and operational today. There is user experience that has been gained and has been factored into the equipment and systems involved, and in the design of the teleconferencing rooms.

Insofar as audio conferencing is concerned, the following is concluded:

- o Audio conferencing systems are in wide use today and audio conferencing is the most universally available, and accepted, form of teleconferencing.
- o Audio conferencing components are readily available at competitive prices and will continue to be for the forseeable future.
- o The most rapid growth in audio conferencing will be in the area of audio-graphics
- o Audio conferencing is the most logical degradation point for a fully developed motion video conferencing system.

Relative to the full motion video teleconferencing capability, the following is concluded:

- o These systems are operational today and are found to be improving the timeliness and quality of communications and decision making.
- o These systems are being accepted as a communications and decision making media by many organizations and the current environment is one in which there is an accelerating rate of the number of organizations which are planning to implement such systems.

- The types of applications and the forms of the information involved should not be untypical of many of the meeting requirements of WWMCCS (development of plans, joint reviewing of presentation or electronically generated material, etc.).
- The increasing utilization planned for these systems will further improve the overall systems. In those areas of equipment which are digitally based , the increased volume production will result in decreased costs.
- The future areas of technical challenge and improvements in cost performance are associated with the codec and the controller which are both digital systems and software based. Multi location capability (a software development) may have to be undertaken if not already developed in the meantime.
- The specific requirements of the government associated with security and survivability can generally be accommodated with proper planning and system implementation and with the acceptable level of operation in a degraded mode.
- A program development initiated in early 1984 with the intermediate step of a two node pilot test, would result in a reasonably paced program resulting in the implementation of a multi-node network by the early 1989 time period.

WWMCCS ADA STUDY: NETWORKING

Prepared for

INSTITUTE FOR DEFENSE ANALYSIS

by

MCQUILLAN CONSULTING

September 13, 1983

Table of Contents

1 Overview

This report presents findings on the implementation of the DoD standard network protocols, using the DoD standard higher level language ADA. In the course of preparing this report, several implementations of these protocols were researched. Case studies are included to provide a hard, concrete basis for estimating the resources required for future protocol implementations.

The software systems we are concerned with here are several:

TCP	Transmission Control Protocol. Provides reliable, sequenced, byte-stream virtual circuits. It features checksums, sequence numbering of all data, retransmissions for reliability, reliable connection establishment and clearing, a window-oriented flow control mechanism, and a mechanism for marking data as urgent.
IP	Internetwork Protocol. Provides TCP with addressing and internetworking capabilities. IP is a datagram protocol that does not guarantee reliable or ordered delivery. Addressing is based on a 32-bit address, part of which identifies the host itself. It provides for fragmentation of message by gateways as they are routed through networks with different message sizes. the fragments are then collected and reassembled by the destination host IP module.
TELNET	Teletype Network Protocol. User TELNET gives the user a remote terminal capability by taking the characters from the local input device and sending them to the foreign host, and returning characters from the foreign host to the local output device (typically a terminal). Server TELNET acts as a pseudo-Teletype, with incoming network messages providing TTY input, and TTY output being sent to the network.
SMTP	Simple Mail Transfer Protocol. Provides for the transfer of electronic messages from one user's "mailbox" to those of the recipients.

FTP	File Transfer Protocol. Allows a user to move files from one computer system to another.

TCP and IP are network transport functions, while TELNET, SMTP, and FTP may be considered applications-level or user functions. The two groups are treated separately below.

The key case studies reported on here include:

- The BBN C-30 TAC: TCP and IP, written in assembly language
- The BBN HP3000: TCP, IP, TELNET, and user FTP, written in SPL
- The BBN VAX UNIX: all protocols, written in C

The typical performance measure applied to network protocol implementations is the maximum throughput level that can be sustained over a given network, in bits/second. Many factors contribute to the actual measured value, including the speed of the network and the test computer. The quality of the software implementation of the protocol can have a major bearing on the network throughput, with a finely-tuned system supporting two or three times the traffic load of a less advanced system, on a comparable set of equipment. Performance may differ by as much as an order of magnitude between computers of roughly comparable price and raw CPU performance, given different hardware and software architectures.

Based on our review of a number of cases, only a few of which are discussed in detail here, we conclude that the implementation effort that has been required in the ARPANET community for these systems is as follows:

TCP, IP	18 man months
TELNET, FTP, SMTP	12 man months

We would estimate a similar amount of effort would be required for future implementations programmed in ADA, given the same set of environmental conditions (see below for details). The use of ADA is seen as neither increasing nor decreasing significantly the amount of time required to develop an operational protocol implementation.

The estimates are based on the following important assumptions, some of which may not hold for some protocol implementations in the WWMCCS environment:

- Programmers experienced in ADA and the target computer
- Programmers with network experience, and familiarity with the DoD protocols
- A model protocol implementation for another computer to use as a guide
- Protocol testing tools such as traffic generators, echoes, and discarding destinations, which follow the appropriate protocols correctly.
- A requirement for informal research-oriented documentation only, as opposed to formal military-specification documentation
- No security considerations such as working in a trusted software environment, or carrying out a verification procedure
- Flexible operating systems, such as UNIX and VMS, with powerful tools for interprocess communications

- Very high-caliber programmers (top 5% in the field) working with the support of other excellent programmers "in the community", with extensive network protocol experience

We speculate that if any of the above factors is adverse, it could easily double the effort required to implement a protocol. If several of the factors are adverse (e.g., an inexperienced programming team working with a difficult operating system in a secure programming environment with stringent documentation standards), the actual effort required would be significantly greater than our basic estimate. It would not be surprising that some implementations might require ten times the effort we are suggesting, under compound adverse programming conditions.

Perhaps more significant than the effort required to implement these protocols is the manpower required to support, maintain, and enhance them. We estimate that 1 to 2 full-time programmers or the equivalent are needed to support each version of TCP/IP running on a different computer or operating system. Note that this estimate is also predicated on a favorable conditions with regard to the documentation, security, operating system, programming staff, etc. A level of effort of 2 programmers is probably sufficient to deal with up to 100 installations of exactly the same type.

2 Discussion of Functional Requirements

The functional requirements for the DoD protocols are documented in working notes and official specifications available from the Network Information Center at SRI International. In addition, there is a continuing on-line newsletter called the TCP-IP Digest which disseminates information within the community of interest.

One of the key insights that has emerged from this experience is that the protocol implementer must, like the protocol designer, make design choices so as to tailor the implementation to users' needs and performance requirements. In a given network, it is possible for the well-implemented "wrong" protocol to out-perform the poorly-implemented "right" choice.

Because the DoD protocols are standards, any implementations that conform to the TCP/IP standards will be able to communicate with each other, but perhaps not optimally. It has been common to bring a protocol into service quickly by using the best available initial implementation, leaving for later the tailoring of a better solution. Often this later work involves extensions in functionality or improvements in performance.

Protocol developers have found a three-way tradeoff between

1. Functionality. The user's capabilities.

2. Computer performance. Efficiency in processing and memory use.

3. Network performance. Efficiency in circuit bandwidth and other network resources.

These three goals are somewhat in conflict, presenting the implementer with a three-way tradeoff, because each implementer, with a limited budget and schedule, must choose which goal to emphasize. A typical example is the choice between adding one more user feature (e.g., monitoring and fault-isolation diagnostics) vs. improving the performance of the existing features. Performance might be improved by adding more buffers (because data must be held in memory until an acknowledgement is received) or by speeding up the processing logic. The former optimizes network performance at the expense of host computer resources. In the latter case, computer performance is optimized. We discuss each of the three factors in greater detail below.

Functionality. The typical experience in implementing TCP and IP, and the recommended course of action, is to begin with a subset of the functionality, and then add advanced features. Basic TCP features include connection management, packet logic, checksums, sequence numbers, basic error control and flow control. Advanced features include adaptive retransmission, TCP options, and improved error control and flow control strategies. Basic IP mechanisms are simple addressing and checksums. Advanced features are aspects such as fragmentation and reassembly, multi-homing, and option handling, and true internetwork gateway addressing.

Computer performance. Performance analysis should be used to determine which sections of code are most frequently invoked. Per-packet overhead can be reduced by the efficient coding of such routines as checksums and byte-swapping (possibly in assembly language). Processor interrupts can be minimized by increasing I/O buffer size, and general performance can be increased by replacing subroutine calls with in-line code, and optimizing frequent code paths.

Network performance. An important technique here is to minimize the number of packets necessary to transfer a given amount of data. Because of TCPs lattitude in packetization, error control and flow control strategies, this factor can vary widely between systems and within a particular implementation, based on network conditions. Refined transmission strategies are not necessary to adhere to the standard, but they can be very important to overall costs. The best solutions have been based on adaptive retransmission algorithms, which delay transmission until enough data has been gathered for an efficient transmission, and avoid unnecessary flow control or error control activity.

3 Case Studies

In choosing the case studies to report on here, we researched the the TCP/IP implementations that are presently operational on the ARPANET. Investigation revealed that the vast majority of the systems fall into a relatively small number of classes:

1. The BBN VAX UNIX implementation, and the Berkeley VAX UNIX and BBN C/70 UNIX systems, both of which were derived from the BBN VAX UNIX system. Various other PDP-11 systems are derived from either the BBN or Berkeley systems. Over 75 installations.

2. The BBN TAC system. Over 50 installations.

3. The BBN TOPS-20 system. Over 25 installations.

Other operational systems include those for the HP3000, the PDP-11 under RSX11M, Univac 1100, the IBM MVS, and MULTICS. We know of no implementations written in ADA, or in PASCAL.

We have chosen to report on three case studies below, the BBN TAC and VAX UNIX systems because of their widespread operational status, and the HP3000 because it was written in a higher level language somewhat comparable to PL/I or ADA.

Name of System BBN C-30 TAC

Developer Mr. Robert Hinden, BBN (617) 497-3757

Description of System The Terminal Access Controller (TAC) is a user TELNET host that supports TCP/IP and NCP host-to-host protocols. It runs in 32K H-316 and 64K C/30 computers. It supports up to 63 terminal ports, and connects to a network via an 1822 host interface. The TAC TCP/IP conforms with RFC791 and RFC793 specifications with the following exceptions:

1. IP options are accepted but ignored.

2. All TCP options except maximum segment size are not accepted.

3. Precedence, security, etc. are ignored.

The TAC also supports Packet core, TAC Monitoring, Internet Control Message Protocol (ICMP), and a subset of the Gateway-Gateway protocols. For more information on the TAC's design, see IEN-166. All major features have been implemented except Class B and C addressing, IP reassembly, and TCP Urgent handling. These will be done in the near future.

Performance The TAC is designed for high performance in the sense of supporting a large number of low-speed terminal devices simultaneously. This protocol implementation can support up to 63 terminals at a mix of speeds (typically 300, 1200, and 9600 bits/second).

Quality This is a fully operational system, in daily use by thousands of users. It is an extremely reliable utility service on the ARPANET and related networks. It has been in operation since mid-1982.

Development Team The software was developed by two programmers full-time over the course of 12 months, but they were also building an NCP for the same computer, which they estimate to have required about half of their effort. Thus the TCP/IP work is estimated to have taken 12 man months over 12 months. The system was up and running in 6 months, and into operation in the field in 12 months. An additional 6 months were spent in tuning and reengineering the system. In total, approximately 18 man months were spent developing a fully operational TCP/IP package.

Development Environment The system was written in assembly language using a cross-assembler on the DEC TOPS-20 operating system. A number of tools have been developed for this purpose over the years at BBN, including debuggers, and source control facilities.

End Use There are 55 TACs installed in various DoD networks around the world. These are fully operational systems supported entirely by BBN.

Comments This implementation demonstrates that programming in assembly language is not a barrier to the development of a reliable, efficient, fully operational TCP system. In fact, it is doubtful if programming this system in ADA would have made any improvement in the final product, or the cost to develop it.

<u>Name of System</u> BBN HP3000

<u>Developer</u> Jack Sax, BBN (617) 497-3867

<u>Description of System</u> The HP3000 TCP code runs under the MPE IV operating system as a special high priority process. It is not a part of the operating system kernel because MPE IV has no kernel. The protocol process includes TCP, IP, 1822 and a new protocol called HDH which allows 1822 messages to be sent over HDLC links. The protocol process has about 8k bytes of code and at least 20k bytes of data depending on the number of buffers allocated. The TCP code is believed to support all features except rubber EOL. The IP code currently supports fragment reassembly but not fragmentation. In addition provisions have been made to allow the IP layer to accept and act on routing and source quench messages. These features will be added sometime this summer. Security and precedence are currently ignored. In addition to TCP, the HP3000 has user and server TELNET as well as user FTP. A server FTP may be added later. A complete description of the implementation software can be found in IEN-167. For further information see BBN Report 4856 (January 1982).

<u>Performance</u> The performance of the system is very poor. With the optimal packaging of data into messages, only 20 Kbs of data can be supported in a self-loop test, under FTP or under TCP alone. Much of this inefficiency is in the HP operating system, which is very slow in all I/O operations. In particular, all network I/O flows through a device driver for the front end which treats the front end as a half duplex device. Many complex software routines are invoked for each transaction, with the result that the TCP software is most often waiting for the device driver to complete. A rough estimate is that this implementation would be

capable of supporting 35 Kbs if the device driver were not a constraint.

Quality This implementation is too slow and too preliminary to be considered more than a beta test site release at this point.

Development Team The software was developed by a team of two people over a period of one and a half years. One person worked on the protocol issues, and the other on the issues related to the HP3000. The team had to learn the HP3000, the programming language, and the protocols. A total of 3 man years were required for the project. Most of the effort, perhaps 80%, was required for TCP and IP. By contrast, TELNET and FTP are almost trivial. In addition, the FTP implementation borrows much of the TELNET code. It is estimated that the effort might have been reduced to the range of 2 to 2.5 man years with a better development environment (see below) and better initial familiarity with the hardware, software, and protocols.

Development Environment The software was written in a higher-level language called SPL, somewhat similar to PL/I. The development environment was extremely limited. Apart from the compiler, there were no software tools at all, not even a debugger. Debugging by means of print statements probably added as much as 20% to the overall development time, as compared to state-of-the-art tools.

End Use It was originally intended that the system be installed at the ARPA office, but contractual difficulties and a change in the version of the HP operating system have delayed installation. There are no operational installations at present.

Comments This implementation was included as a case study because it was developed in a higher level language somewhat akin to PASCAL or PL/I, and thus represents a point of comparison with future implementations in ADA. This experience suggests that performance can be very adversely affected by operating system constraints and programming language limitations. It also shows the cost of implementing with a team that lacks sufficient relevant experience for the task. This cost is two-fold: implementation time and expense, and inefficiencies in run-time operations.

<u>Name of System</u> BBN VAX UNIX

<u>Developer</u> Mr. Robert Gurwitz, BBN (617) 491-1850

<u>Description of System</u> BBN has developed an implementation of TCP/IP for DEC's VAX(TM) family of processors, that runs under the Berkeley 4.1BSD version of UNIX(TM). The development effort was funded by DARPA. Some important features of the BBN VAX TCP/IP are that it runs in the UNIX kernel for enhanced performance, it is a complete implementation of the TCP and IP protocols, and provides facilities for direct user access to the IP and underlying network protocols. The IP module supports checksums, option interpretation, fragmentation and reassembly, extended internet address support, gateway communication with ICMP, and support of multi-homing (multiple interfaces and addresses on the same or different networks). The TCP supports checksums, sequencing, the ability to pass options through to the IP level, and advanced windowing and adaptive retransmission algorithms. Support is also provided for the User Datagram Protocol (UDP). In addition to the TCP/IP software for the VAX, BBN has developed implementations of the TELNET Virtual Terminal Protocol, File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP), for use with TCP. These protocols are operated as user level programs. Also provided are network programming support tools, such as network name/address manipulation libraries, status, tracing, and debugging tools.

Note: An earlier, unrelated TCP/IP implementation was developed to run as a user process in version 6 UNIX, with modifications obtained from BBN for network access. This implementation, of a much lower quality, was completed in 6 man months. It also had much lower performance. IP reassembles fragments into datagrams, but has no separate IP user interface. TCP supports user and server TELNET, echo, discard, internet SMTP mail, and FTP. ICMP generates replies to Echo Requests, and sends Source-Quench when reassembly buffers are full. The system runs on PDP-11/70s and PDP-11/45s running UNIX version 6, with BBN IPC additions. The software was written in C, requiring 25K instruction space, 20K data space. It supports 10 connections (including "listeners"). Unimplemented protocol features include: TCP - Discards out-of-order segments (work in progress to utilize out-of-order segments). IP - Does not handle some options and ICMP messages. See BBN Report No. 4295

Performance Port to port throughput on the same IMP has been measured at 120-130 Kbs, running through the TCP software, but no higher level protocols, and discarding all data. Over a 10 Mbs Ethernet LAN, throughput was measured at 400 Kbs. The Berkeley version of the software, with further performance tuning, runs at 800 Kbs over the same network and the same computer (a DEC VAX 750).

Quality This package has been completely operational since April 1982. The TCP/IP and higher level protocol software are now available direct from BBN. The software is distributed on a tape, containing the sources and binaries for a 4.1BSD UNIX kernel containing the network modifications and the sources and binaries for the higher level protocols and support software. Documentation is provided in the form of a set of UNIX manual pages for the network access device, user programs, and libraries. In addition, a detailed installation document is provided. Device drivers are supplied for the ACC LH/DH-11 IMP interface, the Proteon Associates PRONET Local Network Interface, the ACC IF-11 IMP interface, and the Interlan 10MB Ethernet interface.

Development Team R. Gurwitz was responsible for almost all of the network protocol software of this project (TCP and IP) as his only project for a period of 18 months. The first 12 months were spent in design, programming, integration, and testing. The final 6 months were spent in tuning and refinement of the implementation. The higher level protocols (TELNET, FTP, and SMTP) were ported from previous implementations in 6 man months, and tuned and refined for another 6 man months.

Development Environment The system was developed in C in the UNIX environment. It was implemented internal to the UNIX kernel, which necessitated work on the C debugger to make it function correctly in the kernel. The only significant development tool employed was the RCS system from Purdue for source code control. Some VAX instruction coding was necessary in assembler language.

TCP measurement tools such as a message generator, an echo process, and a message discarding facility were of key importance in testing.

End Use Development began in earnest in September 1980. A first working version was ready in March 1981. First releases were made to beta test sites in November 1981, and the system was operational in April 1982. There are presently 78 UNIX-based TCP systems operational on the ARPANET, 12 of which are BBN C/70s, 21 are PDP-11s, and the balance, 45, are VAXs. Nearly all of these installations can be traced back to the BBN VAX UNIX implementation. BBN presently supports the system with a level of effort equivalent to 1.5 to 2 full time programmers. They keep the implementation up to date, help users ARPANET-wide, deal with usage-sensitive problems, etc.

Comments This is generally considered to be the premier TCP/IP implementation, both in terms of quality and popularity (as measured by number of installations and derived systems). Much of its success can be traced to the design decision to place the software inside the UNIX kernel for enhanced performance.

4 Analysis

It is our conclusion that the time required to implement these protocols on various computers and operating systems does not depend very much on the choice of programming language. In fact, the three systems described above were programmed at about the same time by three groups of programmers, in different languages, on very different computers, and they each took about 1.5 years of effort to complete TCP and IP and another year of effort for the higher level protocols. This is quite a striking result.

This suggests that the development time was much more strongly affected by the programming style at BBN: small teams of extremely talented individuals, with ready access to other experts around the ARPANET community.

It is also worth mentioning that most of the individuals who we spoke with concerning the implementation of these protocols in ADA were quite negative in their reaction. This includes the most of the programmers of the systems described above, as well as other individuals knowledgeable about implementations on other computers (IBM, Univac, etc.). The consensus seemed to be a distrust of ADA as a large, complex programming language not suitable for high-performance communications software. ADA was seen as more suited to development of applications which run in user space, rather than operating system or kernel software.

There was also some consensus that the choice of programming language would have a major effect on performance. By and large, the implementations with the best performance have been written in lower level languages for operating system or kernel environments. Those written in higher level languages as applications programs had very poor performance. This suggests that it may be advisable to design a special protocol-handling kernel in ADA for WIN.

An example of this kind of implementation is the work done at UCLA for the IBM operating system MVS. Bob Braden of UCLA, a very experienced protocol system designer, built his own access method in EXCP using IBM service calls. This is equivalent to, and runs in parallel with, the IBM standard VTAM access method. In user space, he wrote a primitive operating system or kernel called MOS to support process management, queues, memory management, etc. This in turn supports the DoD protocols. This particular implementation, while not a high-performance system, is perhaps the only reasonable way to approach the MVS operating system.

If such a kernel could be written in ADA, and made to be a part of the operating system of the target computers (i.e., given permanent space in memory, priority with the scheduler, efficient inter-process communication, etc.) then the final implementations might represent a good compromise between high performance and good portability.

A related recommendation is that implementations in which much of the protocol software is installed in a front end device are probably to be avoided. It has been our observation and experience that such implementations are suitable only for simple single-user terminals. For major computers serving many users and processes, it seems essential to do much or most of the protocol work in the main computer.

These two conclusions are borne out by the relatively unsuccessful experience in attempting to port the earlier ARPANET protocol NCP from TENEX and UNIX, where it worked well as operating system code in the main computers, to Honeywell GCOS computers, where it did not work well as user code on a system with a large front end.

5 Alternatives to TCP

This report would be incomplete if it did not address the broader question of alternatives to the DoD standard protocol set - TCP, IP, TELNET, SMTP, and FTP. It is the author's understanding that the WWMCCS Program Office has made a decision to go forward with the implementation of these protocols in ADA. It is not our purpose here to question that decision, but rather to present some of the advantages and disadvantages of the DoD protocols (which we will refer to as "TCP" for brevity).

Our overall finding is that the choice of TCP represents a difficult tradeoff in several rather different dimensions. It is certainly a feasible approach, but one that has some serious drawbacks. On the other hand, the other alternatives seem to have significant drawbacks themselves.

Of all of the functional capabilities of the TCP protocol family, by far the most important (and a unique feature of TCP/IP) is its capability to provide for internetwork communication. This means that two host computers on different networks can establish a communications link, possibly passing through several intermediate and dissimilar networks, which provides for error-free high-performance data transmission.

Given the requirement for true internetwork communication, what are the alternatives? At present, and for the last five years or so, three schools of thought have been popular:

"1. All networks are the same." This is the point of view of the PTTs (telephone carriers), which are usually national monopolies. These groups developed the X.25 standard as the interface for connecting hosts to networks, and for defining a "virtual circuit" through the network. The X.75 standard provides for end-to-end virtual circuits passing through two or more X.25 networks. This is internetworking in the eyes of this community. In some respects they are right. More than twenty countries around the world support public X.25 data communications networks interlinked by the X.75 standard. The X.121 standard provides for the numbering of networks and hosts, much as the international telephone system of country codes and area codes has been standardized.

Of course, this standard is not a complete solution. Non-X.25 networks are not included in any direct way. Private networks, local networks, and other "special cases" must be made to appear like an X.25 host to be supported.

It is also important to point out that X.25 is a fairly low-level standard. Much has been made of the ISO 7-level reference model for Open System Interconnection:

Physical Layer	Transmits raw bits over a communications channel
Data Link Layer	Transmits data frames sequentially, processes acknowledgement frames
Network Layer	Accepts messages from the host, transmits packets into the network
Transport Layer	Manages communications from source host to destination host
Session Layer	Establishes end-to-end connections
Presentation Layer	Provides services such as conversion, compression, encryption
Application Layer	User-specific protocols

X.25 is seen as a level-3 standard in this model. There is very little international agreement as yet on the higher levels of protocol. Some encouraging progress is being made on the standardization of electronic mail. This application is a good example of the effort required to develop workable international standards. The message format standard spells out which fields must be present in an electronic message, and which are optional. It also defines their exact syntax and semantics. Standards are also required for naming and addressing (the "envelope" for the message), and for the message handling protocol. This work must be repeated in several other areas as well (file transfer, remote job entry, graphics, etc.) before this set of protocols can be considered complete.

<u>"2. All networks are similar".</u> The second point of view is advocated by IBM and other computer manufacturers, who have developed their own network architectures. IBM's Systems Network Architecture (SNA) is the most prominent example. It is by far the most widely installed network in the world today, with over 10,000 IBM host computer systems connected to thousands of SNA installations. The number is doubling every two years.

Other manufacturers have developed competing architectures, such as Digital Equipment's Digital Network Architecture (DNA), and similar offerings from Burroughs, Honeywell, Univac, etc. Recently, there has been a trend towards the development of compatible hardware and software for interconnection of these vendors' equipment with IBM SNA networks. Also, there are now several dozen vendors of protocol converters of various types, to permit the connection of non-SNA terminals and computers to SNA networks.

Finally, in the last year, IBM has changed its stance significantly on the philosophy behind SNA. Originally, SNA was a "closed" network, available only to IBM customers. IBM recently published many of the protocol specifications for SNA to facilitate third-party interconnection. The firm has also announced its support of a wide variety of communications alternatives, including the use of X.25 packet switching networks. This does not mean that SNA is "compatible with" or "the same as" X.25, but it does mean that an organization with an SNA private network can make use of X.25 public data network links as virtual circuits, much as they would use physical circuits. One of the continuing problems with SNA and the ISO model is that there is no simple correspondence between the SNA protocol levels and the 7 ISO levels.

3. "All networks are different." The third point of view is that held by the designers of the DoD standard protocols, which will we term TCP for short. This is also the view of the academic and research community, who feel that it is essential to develop a protocol family with explicit internetwork capability.

Advantages. TCP has two characteristics which set it apart from X.25 and SNA: First, it permits the construction of a composite network using a variety of different technologies (satellite, radio, local area network, etc.). Second, the TCP virtual circuit remains intact when components of the internetwork fail. Given the existence of multiple paths between two users, the composite network will switch paths as necessary to maintain connectivity. In the X.25/X.75 approach, failures will break the virtual circuit, and alternate paths are initiated only by establishing another connection.

The capability for true internetworking may be important to WWMCCS for several reasons. The architecture for WIS, as we understand it, is inherently a multi-network structure. The DDN architecture is basically an internet comprising the ARPANET, the MILNET, and the C3INET. Finally, it is important to note that the IPLI, presently the only secure gateway, processes only IP datagrams, thus requiring TCP/IP.

TCP's other advantages include the fact that it is operationally proven, is mature at this time, and supports a wide range of user-level protocols. But the major advantage of TCP is that it is independent of hardware vendor. This gives DoD the opportunity to write the software first, and procure hardware later, at more favorable price/performance levels. It also makes it possible to switch to better hardware at any time in the future.

Disadvantages. However, TCP does have its drawbacks. It is an expensive protocol to operate. It uses a great deal of host computer resources, especially on more traditional operating systems ill-suited for high levels of interprocess communications. It is also inefficient in network resources, using very long TCP/IP headers on all packets, even very short interactive traffic. (This may not be that important if future interactive traffic tends to stay local, and pass over high-speed LANs, but it is certainly a concern at present). There is no question that TCP is overkill on X.25 subnetworks, which may dominate in the long run.

Another issue is that TCP is essentially a creature of DoD. It remains to be seen if any other body of users will embrace it. As a result, vendor support for the protocol set is likely to be weak. We have heard that IBM, Univac, and some others are working on TCP, but not as supported products. This means that DoD will probably have to bear the almost the entire cost burden of development, enhancement, maintenance, and support of the TCP protocol family on as many computer systems as necessary.

A final drawback which is potentially more serious is the fact that we have no operational experience to proove that TCP written in ADA can indeed be ported from computer to computer, <u>resulting in a high-quality implementation</u>. Most people familiar with the subject would grant that an ADA implementation is portable in the sense that it could be made to work on another computer. But there are serious reservations in the protocol community about the performance of the resulting system. The TCP implementations reported on here have utterly different philosophies and architectures. The fact that the TCP algorithms are written in a portable language is only half the battle. More experience here would be extremely valuable.

6 Conclusions

There are several strategic issues to be addressed in considering the development of many implementations of the DoD protocols in ADA:

1. The use of ADA facilitates computer independence and portability. There are serious computer and network performance questions to be answered, i.e., do ported ADA implementations perform well? do implementations written in user space as opposed to operating systems kernels perform well?

2. The functionality and performance of TCP, etc. depend very much on the computer and network architecture. What is the WIS architecture, and where will TCP reside?

3. Phased implementation is advised. How will the implementations be phased? What is essential for early implementation?

4. Careful tradeoffs between functionality, computer performance, and network performance are essential. What guidance will the WWMCCS management provide to implementers?

5. There is also a basic tradeoff between the upfront cost to develop the software and the ongoing costs involved in maintaining it, and paying for the computer and network resources consumed. What guidance will the WWMCCS management provide here to implementers?

The choice of ADA as the implementation language for TCP, IP, TELNET, SMTP, and FTP is a difficult one. Indeed, the choice of this protocol set itself has advantages and disadvantages. We cannot comment on the suitability of the approach as a whole without the perspective of the WWMCCS management. We do offer a professional opinion on the risks involved.

It is the conclusion of this study that the major risk in the use of ADA for TCP, etc. is not in the time and cost of initial software implementation. That cost is determined much more by other factors such as staff quality and experience, requirements for documentation and security, and the quality of the operating system and network tools. The key risk is that the implementations will have unacceptably high computer and network operating costs. We regard this as a serious risk and recommend detailed benchmark studies be carried out to assess the cost penalty of a ported ADA system as compared with a "native" system on several common WWMCCS computers. It is only with this kind of hard data that an informed decision can be made on the tradeoff between the computer independence offered by ADA/TCP and the costs involved in gaining that independence.

CLUSTER IV PAPERS

software design & analysis, inc.

1670 Bear Mountain Drive
Boulder, Colorado 80303

303 499 4782

A Plan for Acquiring Aids for Converting Fortran or Cobol to Ada

William E. Riddle

ABSTRACT:

We propose a two-year plan for developing the capability to convert Fortran or Cobol programs into Ada programs. This plan is based on the consideration of two existing capabilities for the conversion of one high-level programming language into another.

This plan was produced as part of a study, performed for the WWMCCS Program Office, to define software and system level building blocks for command and control systems.

software design & analysis

1670 bear mountain drive, boulder, colorado 80303

1 November 1983

Kathy MacDonald
IDA
Arlington, VA.

Dear Kathy,

IDA has my permission to reproduce the two technical papers I prepared as part of the IDA-supported WWMCCS support environment study.

Sincerely,

[illegible]

William E. Riddle, President
for
software design & analysis, inc.

sda

1. Overview

1.1. Brief Description

Conversion Aids are tools that assist in the translation of programs in one high-level programming language, called the source language, into programs in another high-level programming language, called the target language. The aids ideally perform this translation fully automatically, but several technical problems usually result in the aids performing only the bulk of the translation, with human interaction or involvement needed to complete the translation.

This study is concerned with conversion aids where the target language is Ada*. The source languages of interest are primarily Fortran and Cobol because of the large amount of existing code in these languages within the command and control community.

The task of conversion is usually broken into three steps as depicted in Figure 1.

---

* Ada is a registered trademark of the Department of Defense Ada Joint Program Office.

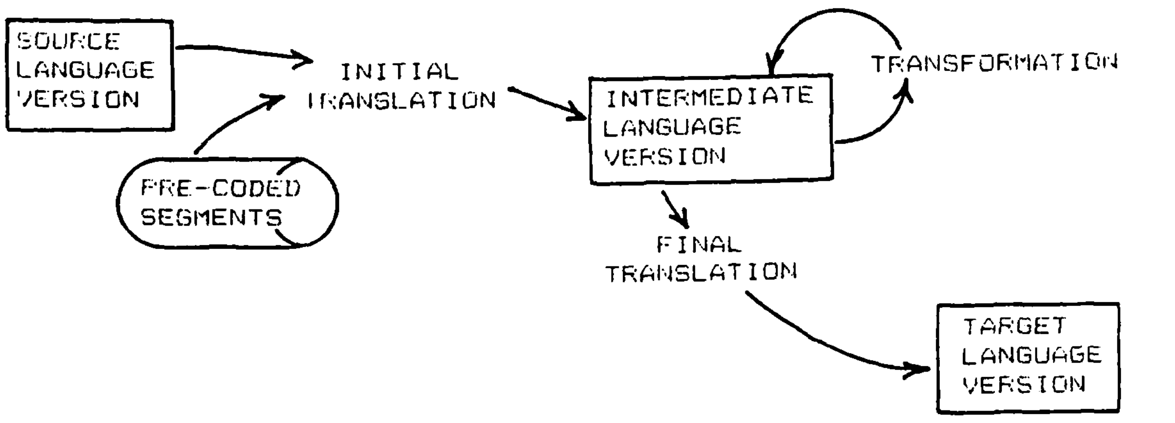


Figure 1: General Organization of Conversions Aids

The first step involves the initial processing of the program in the source language to produce an intermediate language version. This initial translation is done with respect to a library of pre-coded segments of target language descriptions that have been foreseen as typically used in the conversion process. The second step involves massaging this intermediate language version, through the use of various transformers, to improve the quality of the end result. The third step involves the production of the final target language version from the intermediate language version.

Conversion aids are organized into this sequence of processing steps in order to isolate the generally simple task of obtaining some, albeit primitive, translation from the much more difficult task of getting a good translation. The initial translation can handle the task of providing a primitive understanding of the source version and prepare what is, in essence, a rough translation. The transformation step can tune this rough translation in various ways to increase either the performance or the readability of the resulting target version.

An important general characteristic of a conversion aid is the unit of information in the source language that it works on. While the conversion aid could operate on an input description as an essentially monolithic whole, it is generally better to have it recognize some meaningful substructure within the description. Usually this is a program unit such as a subroutine, function or common block. Not only does this tend to keep the performance of the conversion higher, but it also tends to produce more malleable initial translations. This will be discussed further below.

1.2. Examples Studied

Two examples of existing conversion capability were studied. One is the work done in conjunction with the Interactive Bath Improvement System (IBIS) project at the University of Bath in England and the other is a M.Sc. project done in Australia at the University of Tasmania.

The general goal of the IBIS project is to provide a facility supporting the interactive, user-driven improvement of programs, specifically Ada programs. This means that there is a heavy emphasis on the transformation of programs such as is needed in conversion aids. Apparently this project grew out of earlier and continuing work on conversion aids and it is utilizing the previous thinking about transformations needed in conversion. The general objective of the conversion segment of the IBIS project is to provide a facility for converting from any one of a number of source languages to any one of a number of target languages. The primary focus to date, however, has been on the conversion of Fortran to Ada. The Fortran to Ada conversion aid that is being built has the general structure given above -- in fact, this structure is taken from the literature on the Bath conversion aid. Further, the project uses the Diana intermediate language. The conversion tool as it stands now is able to produce an initial translation, perform some simple transformations, and prepare the final translation. Several simplifying assumptions have been made and a few technical problems deferred -- these are discussed later -- but the Bath conversion aid seems to provide both a proof of concept and a conceptual and technical basis on which to progress.

The project at the University of Tasmania focused on the problems associated with trying to discover structure in converting Fortran programs into Pascal programs. The system that was produced is not extensive enough or of high enough quality to be of direct use. But it demonstrates the possibility of doing at least primitive structure determination so that the target version appropriately uses the control structuring constructs of modern languages such as Pascal or Ada. While this capability is not necessarily required, it does tend to provide target versions that are more maintainable (because they are more understandable) and that provide higher performance (because the control logic is not programmed out a primitive way) and these characteristics can be important as explained below.

1.3. Performance

The major performance measure is the speed of the conversion aid, in lines of source language processed per minute (lpm). Both of the cases studied report on the order of 800 lpm for the conversion processing -- the actual rate for the Bath conversion aid is about half this rate because of the overhead introduced by the Diana interface checking, a factor which the Bath people feel can be reduced.

Thus, conversion can be expected to be a lengthy, resource-consuming process. The initial translation can generally be done in time that is proportional to the length of the source description being analyzed. But the transformational processing can require time that is exponentially proportional to the length of the source description, and there is not much hope of speeding this up due to the nature of the processing that must be done.

Another performance measure that can be used for conversion aids is the speed of the target version relative to the source version (with the speed measures appropriately adjusted to reflect the speeds of the machines on which these two versions run). No such figures were reported for the Bath conversion aid, but it was reported that the Pascal code produced by the Tasmania processor ran slightly slower than the original Fortran version. It should also be pointed out, as reported elsewhere, that the experience has been that autom[illegible]ic conversion generally produces code that runs faster than hand-converted code. Exactly what relative performance is needed depends on many factors; for example, the converted code may still provide acceptable response time even though it runs slower. So the specification of requirements for this aspect of performance is highly dependent on the application area for the converted code and cannot be easily specified in absolute terms.

1.4. Level of Effort to Implement

Because of the research nature of the two examples studied, it is hard to predict the level of effort required to produce acceptable production-quality versions. The data reported by the projects suggest that a bottom line figure for the design and implementation of an initial translation and simple transformation capability (that skirts some technical issues) is about one person year with another person year needed for testing. On the other hand, the people at the Bath project have estimated that to produce a Cobol to Ada conversion aid that provides reasonably sophisticated transformational processing would require about 30 person years over an 18 month period at a cost of about $5M. This seems to be more than is needed and the true required effort is probably somewhere in between but close to the high end for any degree of sophistication -- a reasonable plan requiring about 23 person years to get both a Fortran-to-Ada and a Cobol-to-Ada capability is detailed in Section 4.

2. Description of Functional Requirements

What is required of a conversion aid depends on the end-use of the target version. On the one hand, the purpose of conversion could be to merely obtain a runnable version that will not itself be used a basis for evolution of the software system. On the other hand, one may want to obtain a version that will be used for system enhancement and maintenance. The requirements for producing evolvable code are obviously more stringent.

Without the intent to use the converted program as a basis for system evolution, the requirements are:

- -- the conversion aid should accept and produce versions that adhere to some set of standards for the source and target languages,
- -- the conversion aid should be able to completely automatically produce a target version that is runnable with "minimal" human massaging; the current state of the art and the desire for reasonable conversion aid performance prohibit requiring "no human massaging"; we must at least be willing to accept the situation in which the conversion aid produces a target version that produces compiler error messages to draw our attention to situations which could not be handled automatically,
- -- the conversion aid should handle machine dependencies by inserting appropriate code into the target version whenever possible,
- -- the conversion aid should process source versions on a natural program unit basis (e.g., on a subroutine basis),
- -- the conversion aid should produce target versions that perform acceptably; this requirement must be enunciated more exactly for the different types of command and control software; the conversion aid should probably provide interactive transformational processing for performance enhancement so that the human user can direct and guide improvement of the target version with respect to its performance, and
- -- the conversion aid should allow the easy incorporation of new transformations so that the overall system is extensible as new desires are discovered and new technology is developed.

For the production of target versions that can be used as a basis for evolution, there is an additional requirement: the conversion aid should produce readable, understandable code. This basically means that the conversion aid should produce well-structured code that appropriately uses the constructs of the target language. Providing this capability will negatively affect the performance of the conversion aid and invoking this requirement must be done only in conjunction with a careful assessment of the performance required as well as a careful assessment of whether or not the capability is required.

2.1. General Case

With the structure given above, it is possible to first acquire a basic capability and then enhance it as the need arises and the technology permits. The general case can therefore be considered to be a baseline system that:

-- has both standard Fortran or standard Cobol as source languages (it will probably be necessary to accept both Fortran 66 and Fortran 77),

-- has standard Ada as the target language,

-- handles, correctly, all aspects of the source languages with the proviso that it can produce target versions that generate compiler error messages to indicate constructs and situations that it cannot efficiently handle.

-- provides transformational aids for assisting in removing compiler error messages in the target versions and improving the performance of the target version, and

-- provides the ability to easily incorporate new transformations, perhaps providing tools to assist in this process.

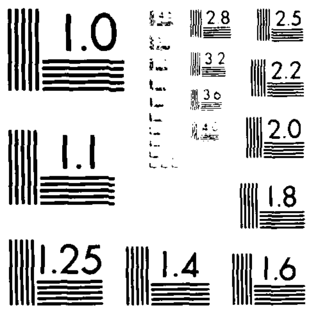
2.2. Variants

The variations that are possible stem from the possibility of adding transformations that assist the user in preparing high(er) quality target versions. An obvious variation is one in which the transformations are provided to produce well-structured target versions. Other variations could include transformations that: produce

AD-A142 570 WIS IMPLEMENTATION STUDY REPORT VOLUME 3 BACKGROUND INFORMATION(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T H PROBERT 01 OCT 83 IDA-D-51-VOL-3 6/6

UNCLASSIFIED IDA/HQ-84-28344 MDA903-79-C-0018 F/G 17/2 NL

END
DATE
FILMED
8-84
DTIC



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

code adhering to some coding standard, produce code that includes the source code as comments in the target version, etc.

2.3. Performance

The basic, general case can be expected to perform linearly with the length of the source description being processed. The transformations can generally be expected to require exponential time.

It seems reasonable to expect that target versions will perform about as well as the source versions, modulo differences in the speeds of the machines on which these different versions run. It does not seem reasonable to expect the target version to perform significantly better. For most applications, it would seem that significant degradation, say by a factor of two, in the performance of the target version will be unacceptable.

2.4. Technical Challenges

For the basic version of the system one technical challenge seems to be handling input/output. The high dependence of this part of a language on the machine taxes the currently developed techniques for conversion. Meeting this challenge certainly does not require advances in technology; it rather requires hard, innovative work on how to efficiently accomplish the needed processing.

The other technical challenge that will be encountered in producing the basic conversion aid lies in finding efficient ways to process intermodule interaction. It was suggested above that the basic system handle source descriptions on a program unit basis. This means that intermodule interactions will have to be handled by message transfer or that the Ada compiler's capabilities will have to be relied upon to sort out situations that are typically handled by linkage editors for Fortran. Figuring out how to appropriately utilize the compiler's capabilities and the message passing concept underlying the Ada language will be difficult. Some relief is obtained by having the basic system do something reasonably acceptable and then providing, as enhancements to the basic system, the transformations that can be used to improve the target code. This would allow the basic system to be acquired in a reasonable period of time and not be delayed by failures to successfully address this technical challenge; but it merely displaces the challenge and does not eliminate it.

Further technical challenges lie in providing the sophisticated transformations that will lead to high-quality target versions. These seem to be algorithm complexity problems rather than algorithm development problems.

3. Case Studies

3.1. Interactive Bath Improvement System

3.1.1. Developer

School of Mathematics
University of Bath
Claverton Down
Bath BA2 7AY
England

Technical Contacts:	John Slape	011 44 225 61244 x 320
	Peter Wallis	011 44 225 61244 x 222

3.1.2. Description of System

The overall intent of the Interactive Bath Improvement System (IBIS) project is to provide an interactive facility for the improvement of Ada programs. As part of this project, work has been done on a conversion aid that translates both Fortran 66 and Fortran 77 into Ada. The conversion aid does not handle all legal Fortran code -- the cases that it does not handle are relatively minor with the exception that it does not handle formated or unformated input/output, a major impact on its utility.

The conversion aid is structured as depicted in Figure 1 and uses Diana as the intermediate language. The initial translation is a line-by-line conversion of Fortran into a Diana tree representation of the Ada target version. This initial translation is done in a single pass, keeping its efficiency relatively high. The transformation processing is fairly simple at this time. The final translation is done by a "pretty printer" that constructs well-formated Ada code from a Diana representation.

The conversion aid produces almost runnable Ada code. In those cases were the translation cannot be automatically done (sometimes because of technological feasibility but primarily because of the one-pass nature of the initial translation), the target version will generate error messages at compile time, so the system can be categorized as one that either succeeds or announces that it has failed. One case in which this is necessary is the handling of Fortran extended-range do-loops for which the generated code will reference labels from outside their scopes. Another is in checking the rule in Ada that a function cannot modify its parameters. The resulting illegal target code can be corrected using transformations and taking this approach keeps the performance of the conversion aid within reason.

3.1.3. Performance

The Bath conversion aid processes source descriptions at about 400 lines per minute of cpu time. About half of this processing is needed for transfer of information through the Diana interface and the project members are attacking this problem and hold out some reasonable hope for reducing this time. Thus the effective processing speed of the conversion aid is about 800 lpm.

3.1.4. Quality

The system is a proof-of-concept prototype.

3.1.5. Development Team

The system as it stands now is about 9000 lines of Pascal code. No information was available about how many people worked over what period of time to produce this code.

3.1.6. Development Environment

Evidently, no special tools other than normal operating system-level tools and the Pascal language processor were used. The system was developed on a Honeywell Multics system but such a system does not provide extensive support for software development other than those aids found in traditional operating systems.

3.1.7. End Use

At the moment, the system runs solely on a Honeywell Multics system, but work is already planned to bring the system up on a Perq work station. The system has evidently not been distributed to other installations.

3.1.8. Comments

This is a system developed as part of an on-going research project. Thus it provides ideas, techniques, a reduction in design time, and a proof of concept but it does not provide either a usable system or usable piece parts.

While there is a standard definition of the Diana intermediate language, several differing implementations have arisen because of the perception that an implementation of the standard will adversely affect compilation time. Thus the reasonable hope that basing a system on Diana will lead to reduction of effort because of the possibility of "borrowing" parts of the system from others may not be realized.

3.1.9. References

J. K. Slape and P. J. L. Wallis. Conversion of Fortran to Ada Using an Intermediate Tree Representation. Tech. Report, School of Math., University of Bath, August 1983.

3.2. Tasmanian Conversion Aid

3.2.1. Developer

R. A. Freak
Department of Information Science

The University of Tasmania
GPO Box 252C Hobart
Tasmania 7001
Australia

Current Contact: A. H. J. Sale 011 61 2 20 2374

3.2.2. Description of System

This system was developed by Freak as part of an M.Sc. thesis project at the University of Tasmania. The focus of the system is on techniques for producing well-structured translation when converting from one high-level programming language to another, in this case from Fortran to Pascal. The system is a research tool that is incomplete, has some remaining errors and is not robust.

The system shows the feasibility of discovering structure in code that does not have the structuring constructs of a modern programming language. It also shows that such processing is expensive because of the graph searching nature of the algorithms. And it shows that the performance of these algorithms depends on whether or not the source description itself is of high quality, structure-wise -- badly organized code just cannot be converted to good target code that uses the available structuring constructs.

The system works at the Fortran subroutine level and discovers (in a reasonably high-quality piece of source description): structured control flow, block nesting, structure within common blocks that can be realized in Pascal records, and the structure within expressions needed to produce equivalent Pascal expressions.

3.2.3. Performance

Experimental use of the system shows that it processes about 800 lines of source description per cpu minute. Because of the exponential nature of the processing that must be done, it seems unlikely that this can be improved significantly.

3.2.4. Quality

This is a proof-of-concept research-oriented system.

3.2.5. Development Team

No information was available as to the person year effort or time span needed to produce the system. It consists of 14,000 lines of Algol code.

3.2.6. Development Environment

The system was programmed in Burroughs 6700 Algol, presumably with no more than the level of development support available in a traditional operating system.

3.2.7. End Use

There has been some distribution of the system but with no support provided.

3.2.8. Comments

This system provides ideas, algorithms, a reduction in design time, and a proof of the feasibility of producing well-structured target descriptions. It does not offer a usable system or usable piece parts.

3.2.9. References

R. A. Freak. A Fortran to Pascal Translator. Software Practice and Experience, Vol. 11, (1981), 717-732.

4. Analysis

The possibility of providing a tool for converting Fortran or Cobol to Ada is certain. And the tool can be of relatively high performance as long as the sophisticated processing needed to provide advanced capabilities is relegated to optionally invoked portions of the system. This can be accomplished by adopting the structure given in Figure 1.

Diana is an obvious choice for the intermediate representation, especially when the conversion tool is going to be part of an environment supporting the development of Ada-based software systems. The augmented tree structure used in the Diana system provides a fairly high degree of flexibility for organizing the information needed for conversion and organizing the transformers that are involved in conversion.

The processing needed to perform quality-enhancing transformations can be resource intensive. The need for these transformations should be carefully determined before an investment is made in acquiring them and using them. The approach embodied in Figure 1 provides the ability to incrementally enhance a base system as the need for additional capabilities is identified. This aspect of the system should be exploited by delaying the addition of sophisticated transformers as much as possible and then gradually introducing the transformers in stages.

A possible sequence and schedule of activities for acquiring Fortran and Cobol to Ada conversion aids is given in Figure 2.

```
:            84            :            85            :
j-------a-------j-------o-------j-------a-------j-------o-------j
:                          :                          :
*-------------------------------------------*         :
:  develop a basic Fortran-to-Ada conversion aid      :
:                          :                          :
*-------------------------------------------*         :
:  develop a basic Cobol-to-Ada conversion aid        :
:                          :                          :
:                          :              *------------*
:                          :  test the conversion aids :
:                          :                          :
:            *------------------------------*         :
:              develop advanced transformers          :
:                          :                          :
```

Figure 2: Conversion Aid Activities and Schedule

The major activities are two parallel efforts to develop Fortran-to-Ada and Cobol-to-Ada conversion aids. There will obviously be a good deal of synergism possible between these two projects and they must be tightly coordinated.

During the initial half-year of these projects, a definitive specification of the intermediate language must be developed. (If Diana is adopted, as suggested above, then this time is needed to identify the Diana version to use and develop any specialized Diana-related tools pertinent to the conversion task.) This initial definition sub-activity is needed not only to help coordinate the two basic projects, but also to provide a base for the parallel activity, starting in mid-year 1984, that is oriented toward developing advanced transformers. The delay of this latter activity is needed because of the need to define the intermediate language but also because of the need to determine exactly what will be in the basic conversion aids before launching the development of advanced transformers. The final activity in this rough plan is to test the conversion aids and the advanced transformers during the last half of 1985. This will be primarily for certification but it will also serve to uncover any performance problems so that they can be corrected before, or soon after, the conversion aids are put into service.

The effort required and appropriate for these activities is estimated to be:

develop Fortran-to-Ada conversion aid	5 person years
develop Cobol-to-Ada conversion aid	5 person years
develop advanced transformers	7 person years
test the conversion aids	6 person years
TOTAL	23 person years

5. Conclusions

The conclusions of this investigation of conversion aids are:

-- the feasibility of developing Fortran and Cobol to Ada conversion tools has been demonstrated and the acquisition of these aids is primarily a systems engineering task.

the system should be structured as in Figure 1 so that it is possible to first provide a basic capability and then enhance it as deemed necessary and desirable,

-- the acquisition of the conversion aids can be accomplished before January 1986 with an expenditure of about 2[illegible] person years at a cost of approximately $2.9M over the next two years,

-- if the conversion aids are to produce versions that can be used as the basis for system evolution, then the performance of the base system may be severly degraded and the effort required for acquisition will be closer to 30 person years, and

-- the technical challenges are not severe and amount to developing a sound, efficient, well-engineered system rather than the advancement of the theoretical base for conversion.

software design & analysis, inc.

1670 Bear Mountain Drive
Boulder, Colorado 80303

303 499 4782

A Plan for Acquiring Design Description and Analysis Tools

William E. Riddle

ABSTRACT:

We propose a four-year plan for acquiring the capability to describe and analyze designs of concurrent, Ada-based systems. The plan is based on putting a basic capability in place within two years and then enhancing it with more powerful behavior analysis capabilities as the need and the technology permit. The plan results from the study of three existing capabilities, two of which use finite state modelling as the basis for their analysis capabilities.

This report was produced as part of a study, performed for the WWMCCS Program Office, to define software and system level building plans for command and control systems.

PRECEDING PAGE BLANK-NOT FILMED

software design & analysis

1670 bear mountain drive, boulder, colorado 80303

1 November 1983

Kathy MacDonald
IDA
Arlington, VA.

Dear Kathy,

IDA has my permission to reproduce the two technical papers I prepared as part of the IDA-supported WWMCCS support environment study.

Sincerely,

[illegible]

William E. Riddle, President
for
software design & analysis, inc.

sda

1. Overview

1.1. Brief Description

Prior to the implementation of a software system in some programming language, the system must be defined and designed. Definition involves the preparation of a specification for the system, stating the requirements on its functionality and performance. Design involves the preparation of a structure for the system as a set of modules (architectural design) and a specification of each module's data storage and manipulation activities (detailed design).

Design description and analysis tools support the architectural and detailed design activities. Underlying a set of such tools is a design language in which designers can describe:

-- a system's hierarchical decomposition into modules,

-- a module's interface through which other modules can invoke the facilities provided by the module,

-- the desired interactions among the modules at some level in the hierarchy, and

-- the data storage and manipulation aspects of each module.

Descriptions in the design language are models of the ultimate system that specify the details of the system's modularity but are only rough blueprints of the modules themselves. The basic reason for design models is to decompose the overall system's requirements into requirements on the parts of the system.

The tools themselves aid in answering questions about the suitability of the design. The basic question is: if the modules operate as proscribed, will the overall system satisfy its requirements? The tools aid addressing this question by helping designers assess the completeness of the design at each level of decomposition and the consistency between the descriptions of the design at different levels. To facilitate this consistency checking, each module is usually described twice. One description, called the module's external description, specifies those properties pertinent to the module's use. The second description, the module's internal description, specifies implementation aspects such as how the module should be composed out of lower-level modules and how it should use these modules to provide the services advertised to its users by the external description.

Design description and analysis tools assist in:

-- preparing these dual descriptions,

-- checking their completeness with respect to methodological rules concerning what must be specified at various points during design, and

-- determining the consistency between the dual descriptions of each module.

A particularly effective approach to giving a module's internal description is by preparing an operational model in some pseudo-programming language. Such a model gives the details pertinent to understanding how the module controls the use of the facilities provided by lower-level modules. With such models, consistency analysis can progress by analyzing the model, using either simulation or analytic techniques, to determine the behavior that it will exhibit and then checking this derived behavioral information against what is said in the external description about the module's behavior*.

---

* Consistency checking is most usually used to check behavioral characteristics. In the remainder of this paper, we use the terms "consistency checking" and "behavioral analysis" as synonymous.

A design description and analysis tool set consists, therefore, of:

-- a design language,

-- description analyzers that help assure that the form of the description is appropriate and adheres to any rules that might be imposed about how designs should be described,

-- completeness analyzers that help assure that guidelines concerning the process of developing a design description are followed, and

-- consistency analyzers that help assure that the behavior of the modules will be suitable and that the overall system will meet its functional and performance requirements (as long as the modules are correctly implemented).

1.2. Examples Studied

Design description and analysis tool sets differ primarily with respect to the power of their consistency checking capabilities. The major visible effect of this difference is the nature of the design language. Tools sets that provide a natural language as the design language do not provide consistency checkers. On the other hand, tools sets based on a formally defined notation usually include powerful consistency checking analyzers.

Three example tools sets, which span the consistency checking spectrum, were studied. As an example of the low-end of the spectrum, the Byron* program development system was examined. Byron provides linguistic extensions to the Ada** programming language for the description of both internal and external aspects of modules. The

---

* Byron is a trademark of Intermetrics, Inc.

** Ada is a trademark of the Department of Defense, Ada Joint Program Office.

extensions are primarily free-form natural-language text organized into key-worded segments that specify different aspects such as pre-conditions upon invoking a module, post-conditions resulting from module invocation, invariants preserved over module invocation, and the algorithm used in implementing a module. Byron does not provide extensive completeness or consistency analysis capabilities. Despite its relative primitiveness, Byron does provide significant support for design and demonstrates that a fairly modest investment can reap a fairly high payoff.

As an example of a design description and analysis tool set that has an intermediate level of power, the tools being developed in the Software Design Techniques (SDT) project at New Mexico Institute were examined. This is a research project focusing on consistency analysis tools that employ finite state analysis techniques. The SDT design notation permits the modelling of a software system's design in terms of a hierarchy of finite-state machine models of the system's modules. The consistency analysis tools can be used to determine all of the state changes effected by a module's operational model as well as to compare these state changes to those specified in a module's external description.

We consider the SDT tools to be of intermediate power since the consistency checking that they perform is inexact. In analyzing a module's internal description, the SDT tools will develop a description of all of the module's possible behavior, but the derived behavioral description will also include some behaviors that are not possible. This "conservative estimate" of the actual behavior results from from the desire to keep the performance of the analyzers fairly high by having them do an imprecise analysis. Since the result is not exact, it must be interpreted by the designers to come to a definitive assessment as to what the behavior is and whether it is acceptable.

The Hierarchical Development Methodology (HDM) was considered as an example of the relatively high-powered end of the spectrum of possible design description and analysis tool sets. HDM also has a finite state conceptual basis, but uses formal verification technology as the basis for consistency checking. HDM is more powerful than SDT in the sense that HDM performs an exact assessment that results in a definite assessment. As a result, HDM's analysis is considerably more complex and its value must be measured by a benefit/pain ratio where the "pain" factor reflects both the resources required to use the tools and the training needed to learn to use them effectively.

1.3. Performance Issues

Design description and analysis tools can be provided as either batch or interactive tools -- Byron is a batch system, whereas SDT and HDM provide interactive facilities. For batch, measures like the number of lines of description processed per minute are appropriate and for interactive use, response time is the appropriate measure. These are essentially the same measure and reflect the complexity of the processing being performed.

The type of analysis done by SDT and HDM is essentially exponentially proportional in time to the description's length. Significantly less computationally complex algorithms do not seem possible.

so the effective use of these tools requires that a system be well modularized not only with respect to system structuring principles but also with respect to the concerns of analysis effectiveness.

Thus, the issue of design description and analysis tool performance is more an issue of artful, experienced usage than inherent performance characteristics of the tools themselves. More comments about this are made below.

1.4. Level of Effort to Implement

The level of effort estimates for the three tool sets examined vary widely: 10-15 person years over a year-and-a-half for Byron, three person years over a year-and-a-half for SDT, and an indeterminate effort (probably in excess of 30 person years over eight years) for HDM. The variance reflects the differing nature of the projects producing these systems more than the inherent difficulty of moving higher in the spectrum of power.

If one assumes that the underlying technology is well-developed, and therefore little research or requirements modification will be necessary, then one can reasonably estimate that moving to the SDT level of power will require about the same effort as obtaining the Byron level of power and that moving to the HDM level from the SDT level will require about one-and-a-half to two times that amount of effort. These estimates include design and coding and assume the use of a software engineering environment such as Unix* or Interlisp.

---

* Unix is a trademark of Bell Laboratories.

2. Discussion of Functional Requirements

A set of design description and analysis tools provides a notation for stating design-level information, tools for analyzing completeness and consistency, and a conceptual basis that integrates the notation and the analyzers.

For design description and analysis tool sets useful in developing Ada-based systems, the requirements for the conceptual basis are:

-- it must be compatible with Ada's conceptual basis, namely that a system is composed of asynchronously operating modules that interact by message transmission,

-- it must allow the description of a system at a higher level of abstraction than that provided by Ada itself, and

-- it must allow expression of design-level characteristics such as modular structure, module interfaces, and module interactions.

Thus the conceptual basis must be more abstract than Ada's and be oriented toward the description of a system's organization and its behavior.

The design description notation must capture this conceptual basis in a usable form that permits automated analysis. The specific requirements are:

-- it must allow the specification of a module from two points of view: a use-oriented view and an implementation view; this is necessary so that a module can be described either with or without attention to its implementation detail.

-- it should utilize Ada's constructs and syntax whenever desirable and possible; in determining desirability, attention should be given to whether a similar or identical notation for design will confuse the distinction between a design and an implementation.

-- it should allow the description of implementation details in an abstract form so that only the details pertinent to design issues need be expressed, and

-- it should allow the rigorous analysis of design descriptions; this means that the notation should be as formal (but easily usable) as possible.

The notation assists in capturing design-level information and the analyzers must assist in answering questions about a design. The analyzers' specific requirements are:

-- they must automate the analysis process as much as possible but recognize the human users' superior capabilities to cope with uncertain or ill-defined information; thus fully automatic analysis is not necessarily an aim, and

-- they must allow the analysis of:

  -- descriptional characteristics: syntax, standard formats, absence of circular definitions, etc.,

  -- organizational characteristics: correct use of interfaces (i.e., correct number and type of arguments), legal references to shared objects, etc., and

  -- behavioral characteristics: suitable interactions among modules, appropriate performance characteristics, etc.

2.1. General Case

A minimal design description and analysis tool set should include a syntax checker and a pretty-printer for the notation. The notation could be an augmentation of the Ada language so that Ada itself could be used in expressing design-level information. But all that is really needed, however, is a notation that is compatible enough with the Ada language that any implementation-level information that might appear in a design (such as an algorithm's control logic) can be automatically derived from a design description.

Additional tools that should be provided in the general case are:

-- analyzers that check Ada inter-module interface rules such as type correspondence,

-- analyzers that check inter-module coupling such as delivery of required objects,

-- analyzers that check the consistency between internal and external descriptions, at the very least in those cases where the analysis can be done by simple "inspection" (e.g., checking that an internal structure can lead to modifying those objects which the module's external description says it might modify),

-- static and dynamic analyzers for any pseudo-programming parts of a design description, and

-- report generators for retrieving and presenting simple forms of design documentation.

2.2. Levels

Many levels of power and sophistication are possible, depending on the extent to which behavioral analysis is automated. At one end of the spectrum would be simulation-style support for essentially manual analysis of the behavioral consistency between internal and external descriptions. In this case, procedural models of a module's internal operation would be "executed" using a simulator and the results compared, by the designers, with the effect that the module's operation is supposed to exhibit as specified in the external description. At the other end of the spectrum is fully automatic certification that the model of a module's internal operation produces all of and only the behavior specified by the module's external description.

One can move from the simulation-based end of this spectrum to its automatic certification-oriented end in several steps. First, one can add tools that aid in the comparison of the simulation results with the external specification. Second, one can add static analysis tools that aid the analysis of the module's operational model. Third, one can provide analyzers that determine conservative estimates of the behaviors described by a model. Finally, one can add analyzers that derive exact descriptions of all possible behaviors. For these last two augmentations, one could also provide aids for comparing the derived behavior descriptions with the behavior descriptions appearing in a module's external description. The first three of these steps are fairly simple, but the fourth one is rather difficult.

2.3. Performance

For most of the tools mentioned above, processing time is essentially linear with the length of a description. For the reasonably sized modules typically found in a well-modularized system, therefore, performance will be quite acceptable.

Behavior analysis techniques can, however, exhibit lengthy response times because of time complexity problems. Of course, when one realizes the extent of processing that is being asked for and the potential value of the result, then one should be accepting of lengthy response time. Using batch rather than interactive tools for behavior analysis will make this response time more palatable. And sufficient training will help designers use the tools in artful (and economical) ways.

The net result is, therefore, that we can expect acceptable performance as long as we restrict our attention to basic tools and reasonable levels of power. Striving for more sophistication will require extensive user training and a careful evaluation of the trade-off between needs and performance.

2.4. Technical Challenges

Producing the base system and the extended capabilities up through conservative behavioral analysis does not involve any technical challenges. All the requisite techniques are well-researched and prototype versions have been prepared and experimentally used. The technical challenges exist with respect to obtaining powerful behavioral analysis capabilities that perform acceptably well. Unfortunately, these challenges are all the more severe for behavioral analysis of concurrent systems.

3. Case Studies

3.1. Byron Program Development Tool Set

3.1.1. Developer

Intermetrics, Inc.
733 Concord Avenue
Cambridge, Massachusetts 02138

Technical Contact:	Mike Gordon	617 661 1840 x 2480
Marketing Contact:	John Pates	617 661 1840

3.1.2. Description of System

The Byron tool set provides the capability of describing detailed designs and implementations of program units, analyzing these descriptions for completeness and simple consistency properties, and generating a variety of documentation reports.

Byron's language is an extended version of ANSI-standard Ada.

The extensions augment Ada's inherent specification capability and allow design-related information to be expressed on a program unit basis. The extensions currently provide the ability to describe: an overview of a program unit; the error messages produced and exceptions raised by a program unit; the assumptions made by a program unit about how it will be invoked and the visible effect of invoking the program unit (i.e., the program unit's pre- and post-conditions); the changes made by a program unit to variables and other objects in its environment; invariant characteristics of types; the algorithm used by a program unit; performance issues and considerations; and miscellaneous notes concerning a program unit. For the most part, information is expressed in natural language -- a Byron description looks, at first glance, to be a well-commented, albeit perhaps abstract, description.

One of the two major Byron tools is an analyzer which: checks for syntactically correct descriptions; enforces Ada's strong type-checking rules; checks the consistency of type definitions among separately compilable modules; builds intermediate language (Diana) descriptions of program units; and checks for legal use of the Byron description extensions. In effect, the analyzer enforces a hierarchical decomposition methodology since a user must specify what development phase he/she is in whenever the Byron analyzer is invoked and the analyzer checks for specific descriptive items depending on the phase.

The other major Byron tool is a documentation generator. This tool uses the intermediate language representation and a document template to produce a report. A template for the CE specification defined by MIL-STD-490 is included and other templates have been prepared for producing type and data dictionaries, program unit descriptions, and project status reports.

3.1.3. Performance

Byron is a batch system. The analyzer typically processes 10-20K lines per minute running on an IBM 3033 within a 1.5 megabyte region. The documentation generator typically produces 1K lines per minute. Both these figures are dependent on the relative mix of Byron directives to Ada commands and the typical situation in this regard is, unfortunately, not known.

3.1.4. Quality

Byron is a fully operational, production-quality system.

3.1.5. Development Team

It is estimated that the Byron tools required 10-15 person years of effort over a one-and-a-half year period of time. Byron uses the front end of the Intermetrics Ada compiler but these effort estimates do not include work on the front end. These effort estimates reflect design, implementation, and testing activities.

3.1.6. Development Environment

The Byron tools (other than the Ada compiler front-end) were programmed in Pascal using PWB running on a PDP 11/70.

3.1.7. End Use

The first installation of the Byron tool set is occurring now (September 1983) on an IBM 370 at GTE Strategic Systems. A version for the DEC VAX is scheduled for availability in the first quarter of 1984. Normal industrial style and level of support and maintenance are provided.

3.1.8. Comments

More extensive analysis capabilities would be possible if the added description capabilities were more formal in nature. For example, the algorithm description text, which is currently now a natural language description, could be a formally defined PDL thereby allowing cross-reference and data flow analysis such as provided by the PDL's marketed by Softool and others. Rather than pursue this direction for enhancing the Byron tool set capabilities, more consideration is being given to adding capabilities which utilize the information already determined by the Ada compiler front-end -- data flow analysis of the Ada program text is one such capability. Consideration is also being given to checking the consistency of design level information against implementation level information; for example, checking that the code can actually raise the exceptions as indication in the external specification.

While the Byron tool set's capabilities are fairly straightforward and their value depends heavily on their experienced use, the tool set does seem to be fairly valuable -- Intermetrics quotes a figure of 40% reduction in documentation costs. Further, the design of

Byron admits extensive growth to more powerful capabilities, particularly through the introduction of formalized design description capabilities and associated analyzers.

3.1.9. References

Ada PDL Developers. Ada Letters, Vol. II, No. 6, May/June 1983.

Michael Gordon. The Byron Program Development Language. Journal of Pascal and Ada, May/June 1983.

3.2. Software Design Techniques Tools Set

3.2.1. Developer

Computer Science Department
New Mexico Institute
Socorro, New Mexico 87801

Contact: Allan M. Stavely 505 835 5127

3.2.2. Description of System

The Software Design Techniques (SDT) research group is developing a set of integrated tools for analyzing the logical properties of a concurrent system's design. The common conceptual basis for all of these tools is finite state modelling: states are used to describe characteristics of a system's modules, state sets are used to describe the characteristics of a system itself, and state transitions are used to non-procedurally describe the effect of invoking a module.

The SDT tools allow the formal modelling of software at the abstract levels of description characteristic of architectural design. Each module at any level in the hierarchy is described as a finite state machine having observable states and offering invokable operations which change the state, with the possible effects of each operation described in terms of the state transition it causes. The module's external description provides a use-oriented description akin to the external description of an abstract data type or the specification of an Ada package.

The decomposition of a system is described by providing internal descriptions for the modules. An internal description specifies implementation-oriented aspects in terms of the composition of a module out of (lower-level) modules and the algorithms controlling the operation of these internal modules needed to effect the operations described as available in the module's external description. The control algorithms are modelled in a pseudo-programming language.

The descriptional capabilities support a top-down elaboration approach to design. The use of operational models provides the often absent capability to analytically check each elaboration for suitability.

The SDT tools include:

-- a parser that checks the syntax and builds a tree-structured intermediate representation,

-- an unparser that prepares a well-formated description from its tree-structured intermediate representation,

-- a module selector that extracts from the description of an entire system the description of those parts pertinent to the analysis of a specified module,

-- a finite state analyzer that determines the state change caused by a control algorithm model, and

-- an event tracer that determines whether or not a specified sequence of activity is possible or not.

These tools allow the designer to answer the question: is a module's internal, implementation-oriented description consistent with its external, use-oriented description? This, in turn, allows the designer to gradually elaborate the details of a system's design with a check, at each step of elaboration, that the previously specified and certified properties of the system are preserved.

3.2.3. Performance

As with any analysis of system behavior, the SDT tools are subject to time and space complexity problems. Non-determinism in the models, which can arise either by having non-deterministic transitions or in the modelling of communication within concurrent systems, can cause the time for analysis to grow exponentially with the size of the model being analyzed. Careful, disciplined use of the SDT notation and analyzers can help to keep analysis time and space requirements within reasonable bounds.

3.2.4. Quality

The SDT tools are available in prototype form.

3.2.5. Development Team

Four people, each working quarter to half time, have worked on the SDT tools over a one-and-a-half year period -- at the outside, an effort of three person years. Their activities concerned design and coding; requirements definition is not included in this effort estimate.

3.2.6. Development Environment

The SDT tools were programmed in Pascal under the Unix operating system running on a VAX 750.

3.2.7. End Use

The SDT tools were first available in early-1983 on the VAX 750. Since then, they have been ported (with about one person-month of effort) to a DEC-20 running the TOPS-20 operating system. Most recently, they have been distributed, but not yet installed on, a VAX 780 with Unix at the University of Massachusetts. Normal university-style support is provided.

The use of the tools has been primarily for evaluation exercises. About 30 people have completed six design exercises each and provided an evaluation of the usability and value of the tools.

3.2.8. Comments

The SDT tools are prototypical and work needs to be done on their user interface and their efficiency before production-quality versions would be available.

The capabilities that they provide, however, are important since they allow designers to animate their designs and obtain valuable insight into how a system's modules interact. Appropriate and effective use of the capabilities provided by the SDT tools will require evaluative use on real development projects.

3.2.9. References

Allan M. Stavely, Annual Progress Report: Analysis Tools for Design-level Assessment of Software Systems. Computer Science Dept., New Mexico Inst., September 1982.

Allan M. Stavely. Introduction to the Project and the Prototype Tools. Computer Science Dept., New Mexico Inst., March 1983.

3.3. Hierarchical Development Methodology

3.3.1. Developer

SRI, International
Menlo Park, California 94025

Contact: Karl Levitt 415 326 4172

3.3.2. Description of System

The Hierarchical Development Methodology (HDM) is an integrated set of languages, tools, concepts and guidelines to aid in developing and verifying large, real-world software systems. In general concept and philosophy, HDM is very similar to SDT. The underlying conceptual basis is finite state machines. A system is described as a hierarchical decomposition of modules, each conceived as a finite state machine and each described by both an external and an internal description. Analysis tools allow the checking of the consistency of these two descriptions and can be used to support the reasoned hierarchical elaboration of a system's design.

The HDM analyzers utilize formal verification technology. A module's descriptions are analyzed to formulate theorems and the proof of these theorems constitutes a demonstration that the descriptions are consistent. The appropriate and effective use of this approach of course demands a fairly high level of computer science training.

The HDM tools include:

-- specification checkers that analyze the syntax of descriptions and perform simple, static analysis of consistency among module specifications,

-- verifiers that prove the consistency between a module's internal description (modelled in either Modula or Pascal) and its external description, and

-- a multi-level security verifier that uses the HDM specification and analysis capabilities to prove that the information flow among modules adheres to security restrictions.

None of these tools incorporates the capability to analyze concurrent systems since the technology for the formal verification of concurrency is not yet mature.

3.3.3. Performance

HDM is an interactive system. Response time, the appropriate measure for such a system, is generally acceptable with very few instances of debilitating delay. Of course, formal verification is subject to time complexity problems and the delays can be long because of this. But, using HDM in an artful way can frequently keep these delays within acceptable limits.

3.3.4. Quality

HDM is a transferable, production-quality system.

3.3.5. Development Team

The HDM project started in the early 1970's and throughout the period since then has been both a research and development project. It is extremely difficult to separate out the effort required for defining, designing and implementing the HDM tools as opposed to the effort needed to research and develop the technology. Meaningful effort estimates were not obtained.

3.3.6. Development Environment

The current version of HDM was programmed using the Interlisp-10 system. Extensive tool support was therefore available for the programming task.

3.3.7. End Use

The tools described above were implemented over the period from the mid-1970's to the early-1980's. They are installed at SRI, International, and available for external use through the Arpanet. A project is just now beginning to re-design and re-implement the system in Maclisp for installation at an external site.

3.3.8. Comments

It is not necessarily a good idea to provide verification technology directly to practitioners because its effective and appropriate use demands extensive training and experience. While practitioners will always have to prepare the information needed for verification, it is perhaps best to provide special, designer-oriented languages for this task and then have a central service organization, with trained personnel who translate the designer-prepared descriptions as necessary, perform the analysis, and interpret the results for the designers.

It is also not clear how quickly verification technology will be able to handle the more difficult situations, like concurrency, that arise in modern programming languages such as Ada. The definition of

Ada was affected by the desire to perform formal verification, but the technology must be considerably improved before full formal verification of typical Ada-based systems can be accomplished.

3.3.9. References

B. A. Silverberg. An Overview of the SRI Hierarchical Development Methodology. In Software Engineering Environments, ed. Hunke, North-Holland, 1981.

4. Analysis

The state of progress in developing design description and analysis tools is such that we can reasonably expect, in the near to medium term, to acquire a tool set that:

- -- is based on a language which:
  - -- is compatible with Ada, and
  - -- allows the formal description of design-level information,
- -- provides extensive description analysis capabilities, including syntax analysis, pretty printing, and report generation,
- -- provides extensive completeness analysis, and
- -- provides a medium-level of consistency analysis.

We can expect to provide consistency analysis that either performs simple checks not requiring the derivation of behavior descriptions or performs a conservative behavior analysis such as that done by SDT. We cannot, in the near or medium term, expect to provide consistency analysis for Ada-related designs that are based on an exact analysis of behavior such as done by HDM.

The three case studies suggest that there is a natural growth plan for first acquiring a basic set of design description and analysis tools and then maturing the tool set with gradually more powerful consistency checking capabilities. The initial tool set can be based on a design language that is essentially an augmentation of Ada with key-worded natural language descriptions of design-level characteristics. The descriptional and completeness analysis capabilities provided by the basic tool set can be extended fairly easily by working at the macroscopic level and processing a key-worded segment as a unit rather than processing the information within the segment. Consistency checking capabilities can be added in parallel by formalizing the notations used within a description segment and providing analyzers for these formal descriptions.

Finite state modelling is a reasonable basis for extending the consistency checking capabilities. It is compatible with Ada's conceptual basis. Ada could be used as the basis for operational models, opening the possibility of using Ada-based simulation as a first consistency analysis capability. More extensive analysis could be provided by then incorporating the SDT analysis techniques. Finally, in the long term, the HDM techniques could be added to provide the capability to employ formal verification in design analysis.

Figure 1 charts a set of activities which follow this approach. The first step is to acquire Byron. The second activity is to enhance Byron with additional descriptional and completeness analyzers as well as a simulation-based consistency checking capability. The third activity, overlapping the second, is to re-implement the SDT consistency checking capabilities in the context of Ada and Byron. The final activity is to incorporate the HDM capabilities by first absorbing the HDM specification capabilities, in the first half year of this activity, and then absorbing HDM's current capability to formally analyze non-concurrent systems.

```
:     84     :     85     :     86     :     87     :     88     :
j--a--j--o--j--a--j--o--j--a--j--o--j--a--j--o--j--a--j--o--j
:           :           :           :           :           :
*-*         :           :           :           :           :
: acquire Byron         :           :           :           :
:           :           :           :           :           :
*-----*     :           :           :           :           :
: enhance Byron's descriptional and completeness analysis   :
: capabilities          :           :           :           :
:           :           :           :           :           :
*-----------------------*           :           :           :
: incorporate SDT's consistency analysis capabilities       :
:           :           :           :           :           :
:           *-----------------------------------*           :
:           : incorporate HDM's capabilities    :           :
:           :           :           :           :           :
```

Figure 1: Activities for Acquiring Design Description and Analysis Tool Sets

The efforts that can be reasonably expected for these activities are:

acquire Byron	1 person month	$ 85,000*
enhance Byron	6 person years	750,000
incorporate SDT	13 person years	1,625,000
incorporate HDM	23 person years	2,875,000
TOTALS	42.08 person years	$ 5,335,000

5. Conclusions

The conclusions of this investigation of design description and analysis tools sets are:

---

* This figure includes the Byron purchase price of $75,000.

-- a basic capability can be obtained by purchasing and enhancing Byron; this will require about 6 months and about $835K.

-- the enhancement of Byron can include an Ada-based simulation facility so that operational models of a system's modules can be analyzed for their consistency with external descriptions of the modules' intended behaviors by simulation-assisted "manual" analysis.

-- the enhanced Byron system will provide a basis for extending the consistency checking capabilities of the system.

-- finite state modelling and analysis can be used as the basis for this extension.

-- the SDT finite state modelling and (conservative) behavioral analysis capabilities can be incorporated before January 1986 at a cost of about $1625K, and

-- the system can also include the HDM formal verification-based analysis capabilities but these cannot be in place by January 1986:

-- the HDM formal specification capabilities can be in place by the beginning of 1986 and this would allow design descriptions to include formal definitions of a module's externally visible behavior.

-- the formal verification techniques would be in place about two years later, and

-- it is not presently clear how well the formal specification and verification techniques can handle advanced concepts, particularly concurrency.